Instituto Politécnico Nacional

UNIDA PROFESIONAL INTERDISCIPLINARIA EN
INGENIERÍA Y TEGNOLOGÍAS AVANZADAS

Trabajo Terminal II

“RUEDA FRONTAL DE BICICLETA GENERADORA DE ENERGÍA ELÉCTRICA”

Que para obtener el título de

“Ingeniero en Mecatrónica”

PRESENTAN:

Arias García Luis Donaldo

Cervantes Rodríguez Diego

Hisiquio Santiago Víctor

ASESORES:

M. en C. David Arturo Gutiérrez Begovich

M. en C. Alfonso Campos Vázquez

Dr. Juan Manuel Peza Tapia

Enero 2022

upiita-ipn

Instituto Politécnico Nacional

UNIDA PROFESIONAL INTERDISCIPLINARIA EN
INGENIERÍA Y TEGNOLOGÍAS AVANZADAS

Trabajo Terminal II

"RUEDA FRONTAL DE BICICLETA GENERADORA DE ENERGÍA ELÉCTRICA"

Que para obtener el título de

"Ingeniero en Mecatrónica"

PRESENTAN:

Arias García
Luis Donaldo

Cervantes Rodríguez
Diego

Hisiquio Santiago
Victor

ASESORES:

M. en C. David Arturo
Gutiérrez Begovich

M. en. C. Alfonso
Campos Vázquez

Dr. Juan Manuel
Peza Tapia

Presidente de jurado

Dr. Rafael Trovamala
Landa

Profesor Titular

Dr. Víctor Darío
Cuervo Pinto

upiita-ipn

upiita·ipn

Autorización de uso de obra

Instituto Politécnico Nacional

P r e s e n t e

Bajo protesta de decir verdad el que suscribe Diego Cervantes Rodríguez (se anexa copia simple de identificación oficial), manifiesto ser autor (a) y titular de los derechos morales y patrimoniales de la obra titulada Rueda Frontal de Bicicleta Generadora de Energía Eléctrica

_____, en adelante "La Tesis" y de la cual se adjunta copia, por lo que por medio del presente y con fundamento en el artículo 27 fracción II, inciso b) de la Ley Federal del Derecho de Autor, otorgo a el Instituto Politécnico Nacional, en adelante El IPN, autorización no exclusiva para comunicar y exhibir públicamente total o parcialmente en medios digitales, Plataforma de la Dirección de Bibliotecas del IPN y/o consulta directa en la Coordinación de Biblioteca de la UPIITA "La Tesis" por un periodo de 5 años contado a partir de la fecha de la presente autorización, dicho periodo se renovará automáticamente en caso de no dar aviso expreso a "El IPN" de su terminación.

En virtud de lo anterior, "El IPN" deberá reconocer en todo momento mi calidad de autor de "La Tesis". Adicionalmente, y en mi calidad de autor y titular de los derechos morales y patrimoniales de "La Tesis", manifiesto que la misma es original y que la presente autorización no contraviene ninguna otorgada por el suscrito respecto de "La Tesis", por lo que deslindo de toda responsabilidad a El IPN en caso de que el contenido de "La Tesis" o la autorización concedida afecte o viole derechos autorales, industriales, secretos industriales, convenios o contratos de confidencialidad o en general cualquier derecho de propiedad intelectual de terceros y asumo las consecuencias legales y económicas de cualquier demanda o reclamación que puedan derivarse del caso.

Ciudad de México, a 31 de mayo de 2022

Atentamente

_____

upiita·ipn

Autorización de uso de obra

Instituto Politécnico Nacional

P r e s e n t e

Bajo protesta de decir verdad el que suscribe Victor Hisiquio Santiago (se anexa copia simple de identificación oficial), manifiesto ser autor (a) y titular de los derechos morales y patrimoniales de la obra titulada Rueda Frontal de Bicicleta Generadora de Energía Eléctrica

_____, en adelante "La Tesis" y de la cual se adjunta copia, por lo que por medio del presente y con fundamento en el artículo 27 fracción II, inciso b) de la Ley Federal del Derecho de Autor, otorgo a el Instituto Politécnico Nacional, en adelante El IPN, autorización no exclusiva para comunicar y exhibir públicamente total o parcialmente en medios digitales, Plataforma de la Dirección de Bibliotecas del IPN y/o consulta directa en la Coordinación de Biblioteca de la UPIITA "La Tesis" por un periodo de 5 años contado a partir de la fecha de la presente autorización, dicho periodo se renovará automáticamente en caso de no dar aviso expreso a "El IPN" de su terminación.

En virtud de lo anterior, "El IPN" deberá reconocer en todo momento mi calidad de autor de "La Tesis". Adicionalmente, y en mi calidad de autor y titular de los derechos morales y patrimoniales de "La Tesis", manifiesto que la misma es original y que la presente autorización no contraviene ninguna otorgada por el suscrito respecto de "La Tesis", por lo que deslindo de toda responsabilidad a El IPN en caso de que el contenido de "La Tesis" o la autorización concedida afecte o viole derechos autorales, industriales, secretos industriales, convenios o contratos de confidencialidad o en general cualquier derecho de propiedad intelectual de terceros y asumo las consecuencias legales y económicas de cualquier demanda o reclamación que puedan derivarse del caso.

Ciudad de México, a 31 de mayo de 2022

Atentamente

Victor

_____

upiita·ipn

Autorización de uso de obra

Instituto Politécnico Nacional

P r e s e n t e

Bajo protesta de decir verdad el que suscribe Luis Donaldo Arias García (se anexa copia simple de identificación oficial), manifiesto ser autor (a) y titular de los derechos morales y patrimoniales de la obra titulada Rueda Frontal de Bicicleta Generadora de Energía Eléctrica, en adelante "La Tesis" y de la cual se adjunta copia, por lo que por medio del presente y con fundamento en el artículo 27 fracción II, inciso b) de la Ley Federal del Derecho de Autor, otorgo a el Instituto Politécnico Nacional, en adelante El IPN, autorización no exclusiva para comunicar y exhibir públicamente total o parcialmente en medios digitales, Plataforma de la Dirección de Bibliotecas del IPN y/o consulta directa en la Coordinación de Biblioteca de la UPIITA "La Tesis" por un periodo de 5 años contado a partir de la fecha de la presente autorización, dicho periodo se renovará automáticamente en caso de no dar aviso expreso a "El IPN" de su terminación.

En virtud de lo anterior, "El IPN" deberá reconocer en todo momento mi calidad de autor de "La Tesis". Adicionalmente, y en mi calidad de autor y titular de los derechos morales y patrimoniales de "La Tesis", manifiesto que la misma es original y que la presente autorización no contraviene ninguna otorgada por el suscrito respecto de "La Tesis", por lo que deslindo de toda responsabilidad a El IPN en caso de que el contenido de "La Tesis" o la autorización concedida afecte o viole derechos autorales, industriales, secretos industriales, convenios o contratos de confidencialidad o en general cualquier derecho de propiedad intelectual de terceros y asumo las consecuencias legales y económicas de cualquier demanda o reclamación que puedan derivarse del caso.

Ciudad de México, a 31 de mayo de 2022

Atentamente

Índice

Índice de figuras

Índice de tablas

Lista de abreviaturas

IoT.	Internet of Things.
ISO.	International Organization for Standardization.
ERTRO.	European Tire and Rim Technical Organization.
API.	Application Programming Interface.
CD o CC.	Corriente directa o continua.
CA.	Corriente alterna.
CC-CC.	Corriente directa a corriente directa.
CFE.	Comisión Federal de Electricidad.
S.A. de C.V.	Sociedad Anónima de Capital Variable.
FBS.	Functional Breakdown Structure.
IDEF0.	Integration Definition for Function Modelling.

Lista de símbolos

Wh.	Watt hora.
W.	Watt.
Kg.	Kilogramo.
V.	Volts.
A.	Amperes.
rpm.	Revoluciones por minuto.
mAh.	Miliampere hora.
Ah	Ampere hora.
dm.	Decímetro.
Tind.	Par inducido en el rotor.
k.	Constante de par de motor.
Br.	Campo magnético del rotor.
Bs.	Campo magnético del estator.
Km/h.	Kilómetros por hora.
mΩ.	Mili Ohm.
Hz.	Hertz.
“	Pulgadas.
dB	Decibelio.

Resumen

El presente proyecto consiste en diseñar una rueda frontal de bicicleta de 26 pulgadas generadora de energía eléctrica que pueda alimentar una batería portátil con una capacidad máxima de 5,000 mAh por medio de una salida de 5V/2A o una batería de bicicleta eléctrica con capacidad máxima de 10,000 mAh proporcionando una salida de 54.6V/2A, se podrá alimentar una batería a la vez.

El sistema integra uno o más generadores eléctricos para aumentar la tensión suministrada a la batería con el fin de cargarla sin dañarla, una placa de circuito impreso que acondicione la señal de los generadores y establezca una comunicación Bluetooth con un dispositivo móvil y una aplicación Android que permita el control de la energía entregada a la batería y monitoree la distancia, tiempo, velocidad promedio y ruta recorrida por el usuario.

El propósito del proyecto es no afectar la estructura o funcionamiento de la bicicleta a donde se adapte, proporcionando una solución multiplataforma, elegante y funcional. Además, debido a que el dispositivo se encuentra conectado a la red por medio de la aplicación Android, se integra al movimiento del Internet de las cosas.

Abstract

The present project consists of designing a 26-inch bicycle front wheel that generates electrical energy & can power a portable battery with a maximum capacity of 5,000 mAh through an output of 5V/2A or an electric bicycle’s battery with a maximum capacity of 10,000. mAh by providing an output of 54.6V/2A, one battery can be powered at a time.

The system integrates one or more electric generators to increase the voltage supplied to the battery to charge it without damaging it, a printed circuit board for generator’s signal conditioning & establishing Bluetooth communication with a mobile device, and an Android application that allows the control of the energy delivered to the battery & monitors the distance, time, average speed, and route traveled by the user.

The project's purpose is not to affect the structure or function of the bicycle where it's adapted, providing a multiplatform, elegant, and functional solution. Also, because the device is connected to the web through the Android application, it integrates to the IoT (Internet of Things) movement.

Palabras clave: Rueda de bicicleta, energía eléctrica, batería portátil, batería de bicicleta eléctrica, monitoreo.

Keywords: Bicycle’s wheel, electrical energy, portable battery, electrical bicycle’s battery, monitoring.

CAPÍTULO I. INTRODUCCIÓN

a. Marco de referencia

En el marco de referencia se indican las bases teóricas e históricas del proyecto.

a.1. Histórico

Actualmente, la bicicleta mecánica o eléctrica es uno de los medios de transporte más utilizados en todo el planeta, ya que es ecológica y proporciona un ahorro económico considerable si se compara con otros medios de transportes habituales. Su uso aumenta exponencialmente, uno de los principales factores es la adaptación del vial urbano a este medio de transporte, que gana más adeptos cada día, incluyendo la incorporación de servicios de bicicletas compartidas en grandes urbes como la Cuidad de México [1].

A diferencia de una bicicleta convencional, las bicicletas eléctricas, tienen incorporado un motor eléctrico ubicado usualmente en el eje de la rueda trasera, pero no solo las bicicletas eléctricas han sido revolucionarias, se ha innovado también las mismas ruedas, que ahora incluyen todo el sistema de generación y almacenamiento de energía dentro de sí mismas para poder convertir cualquier bicicleta mecánica en una eléctrica [2].

En general, la popularidad y el éxito de los principales servicios de bicicletas compartidas existentes en el centro de la Ciudad de México, como ECOBICI, Mobike, Ofo y VBike [3] ha ido en aumento al pasar los años desde el 2010 [4] (figura 1).

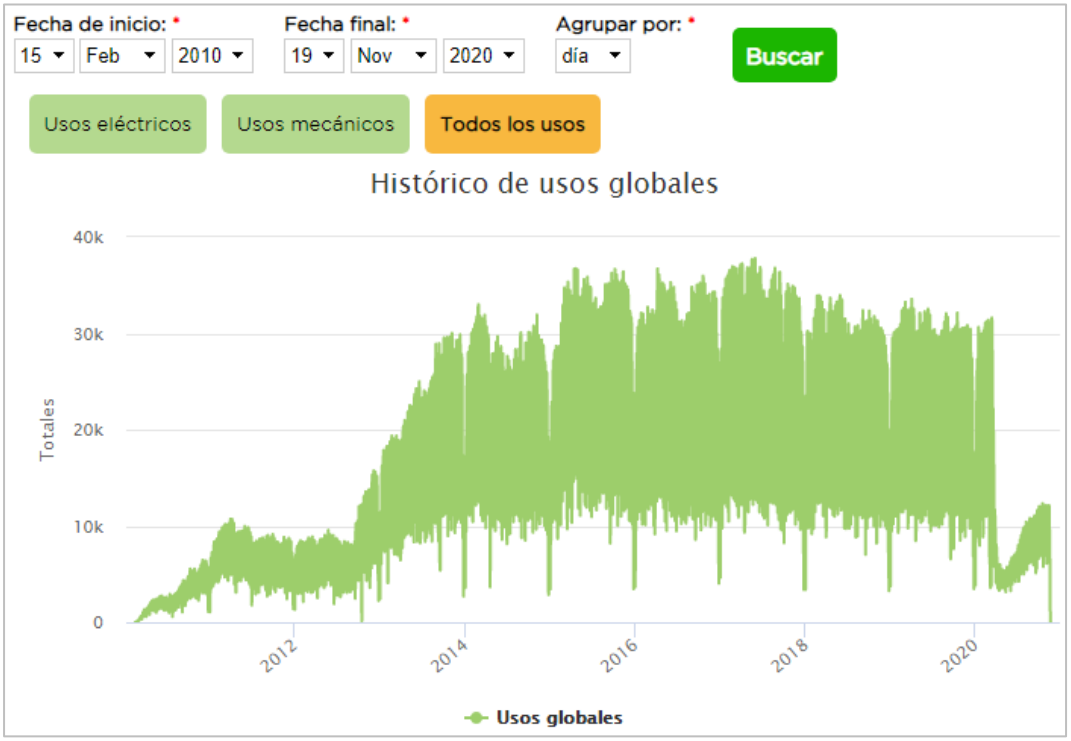


Figura 1. Estadística usos globales ECOBICI [5].

Debido a la alta demanda en los sistemas de bicicletas compartidas previamente mencionado, los autores del presente trabajo, patrocinados por la compañía GRUPO MONTES DE OCA Y DE LA HOZ S.A. de C.V., proponen una alternativa electromecánica situada dentro del rin de una rueda frontal de

bicicleta que genere y almacene energía eléctrica, integrando uno o más generadores para cargar una batería portable o la batería de una bicicleta eléctrica.

a.2. Teórico

A continuación, se presentan las bases teóricas para comprender los distintos aspectos del proyecto.

a.2.1. Rueda de bicicleta

La rueda de bicicleta se compone de un neumático, una cámara de aire, un rin, un buje y radios.

Dentro del neumático se encuentra la cámara de aire, ambos hechos de caucho, el neumático está montado sobre un marco metálico llamado rin, el buje de la rueda es su eje de rotación y los radios conectan al buje con el rin.

Las bicicletas usualmente incluyen 36 radios, aunque para lograr que la rueda sea más ligera, se usan rines y bujes de 32 radios, y para una bicicleta de reparto que está diseñada específicamente para transportar cargas, se usan ruedas de desde 40 hasta 48 radios. Los radios se pueden fijar radial o tangencialmente al buje.

Cuanto más ligeras son las ruedas, más rápido es el manejo de una bicicleta y mayor su aceleración. Por tanto, las bicicletas de carretera y las de pista tienen ruedas ligeras y finas con neumáticos estrechos, mientras que las bicicletas de montaña que se utilizan para saltar rocas y depresiones del terreno usan ruedas más anchas y pesadas con neumáticos robustos. Existen muchos matices intermedios en este espectro y según el tipo de ruedas que utilice, una bicicleta puede tener características diferentes [11].

Partes que integran una rueda de bicicleta:

Los siguientes elementos conforman la estructura rígida de las ruedas (figuras 2 y 3):

Rin. Es el marco metálico sobre el cual se asienta el neumático de la bicicleta, en México a la llanta se le conoce comúnmente como rin [14].

Buje. Es el eje de la rueda de bicicleta. Esta parte central de la rueda puede ser de una sola pieza y se encarga de enlazar los radios con el rin, aportando la unión necesaria entre todos los elementos de la rueda. Usualmente con el buje se incluye el sistema de fijación que sirve para colocar las ruedas en la bicicleta que puede ser por medio de tuercas o un eje externo de cierre rápido que funciona por medio de resortes [15].

Radios o rayos. Son los alambres que unen al buje con el rin de la rueda. Tienen como función principal soportar fuerzas de compresión y flexión para que la rueda pueda girar al mismo tiempo que soporta el peso del ciclista [14].

Neumático o cubierta. Es una pieza toroidal de caucho que se coloca en las ruedas de diversos vehículos y máquinas. Su función principal es permitir un contacto adecuado por adherencia y fricción con el

pavimento, posibilitando el arranque, frenado y manejo; además de proteger a la cámara que tiene dentro [15].

Cámara de aire. Los neumáticos vienen con cámara y sin cámara. Los que son sin cámara presentan un caucho especial en la parte interna, denominado forro, que garantiza la retención del aire.

Las ruedas de bicicleta tienen diferentes medidas, pero están estandarizadas por la Organización Internacional de Normalización (o por sus siglas en inglés, ISO) y técnicos de la Organización Técnica Europea de Neumáticos y Llantas (o por sus siglas en inglés, ETRTO), quienes definen un sistema moderno e inequívoco de designaciones y procedimientos de tamaño de medición para diferentes tipos de neumáticos y rines en la norma internacional ISO 5775 [15].

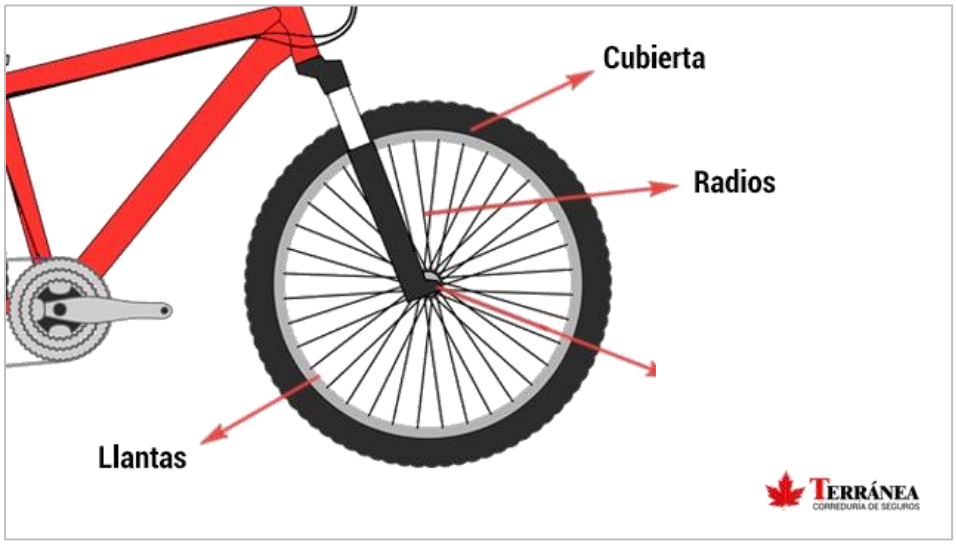


Figura 2. Cubierta, buje, rin y radios de una bicicleta [14].

Horquilla o tijera. Se refiere al brazo de la bicicleta que sujeta y soporta el buje de la rueda frontal.

Figura 3. Horquilla de una bicicleta [14].

Tamaños de rueda. Dado que los tamaños de las ruedas están estandarizados, a continuación, se muestran las siguientes dos tablas con los tamaños de ruedas disponibles en el mercado (figura 4), incluyendo su código francés y su punto de anclaje del neumático.

A continuación, se muestra la tabla 1, que muestra las denominaciones para ruedas de bicicleta de 26 pulgadas de diámetro exterior.

Tabla 1. Rueda tamaño 26 pulgadas [15].

Tamaño	Código francés	ISO	Aplicación
$26x1\,^{3}/_{8}$	650A	590mm	Bicicletas inglesas de 3 velocidades
$26x1\,^{1}/_{2}$	650B	584mm	Bicicletas urbanas
$26x1\,^{3}/_{4}$	650C	571mm	Bicicletas chicas para carreras

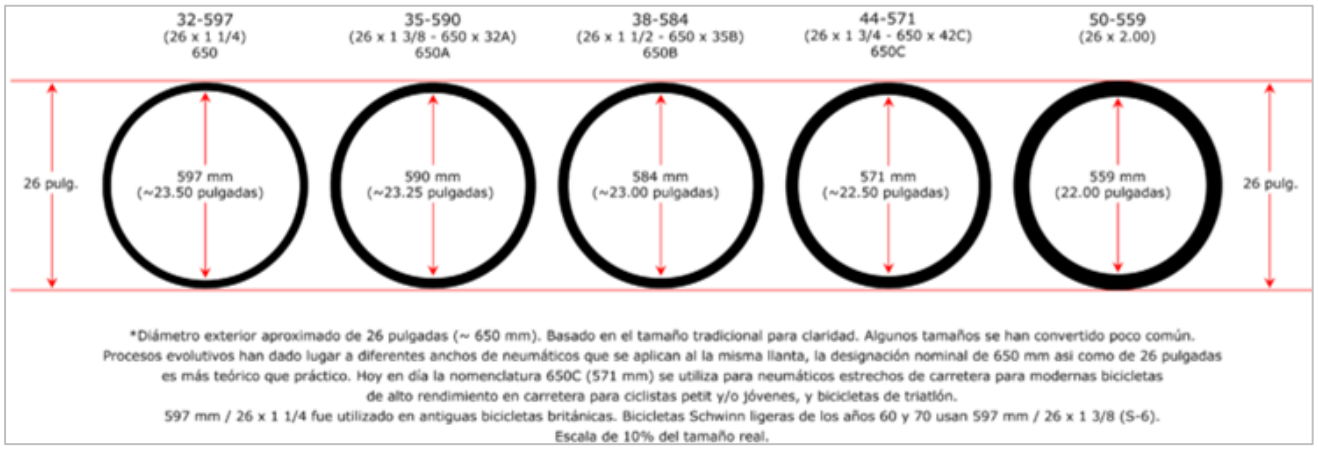


Figura 4. Diámetros interiores y exteriores de la rueda de 26 pulgadas [15].

a.2.2. Internet de las cosas

El movimiento del internet de las cosas (del inglés, IoT o Internet of Things) ha agarrado gran fuerza en los últimos años e impulsa a que se creen dispositivos electrónicos cuyo control o manejo sea por medio de un dispositivo móvil que esté conectado a la red, es decir a un servidor remoto.

Es un concepto que caracteriza la próxima gran transformación en la evolución del internet, su expansión va más allá de la comunicación entre las personas, o entre las personas y el contenido digital; que ahora se extiende a miles de millones de objetos cotidianos, por lo cual se le está denominando como uno de los agentes de la cuarta revolución industrial [16].

a.2.3. Interfaz de programación de aplicaciones

Sus siglas significan Interfaz de Programación de Aplicaciones (o del inglés, Application Programming Interface) y se refiere a una biblioteca de programas previamente hechos en cualquier lenguaje de programación, para que cualquier persona las pueda utilizar o proporcionen datos de todo tipo. Las APIs normalmente no muestran cómo realizan sus funciones o la manera en la que obtienen sus datos, simplemente es observable su resultado, no su código fuente y es muy utilizado para compartir funcionalidades de empresas grandes como Google, Facebook, Nasa, etc.

En este trabajo terminal se utilizará tentativamente la API de Google Maps para mostrar la ruta del usuario.

a.2.4. Generador síncrono

Los generadores síncronos o alternadores son máquinas síncronas que se utilizan para convertir potencia mecánica en potencia eléctrica de CA (figura 5) [17].

En un generador síncrono se produce un campo magnético en el rotor ya sea mediante el diseño de éste como un imán permanente o mediante la aplicación de una corriente de CD o corriente directa a su devanado para crear un electroimán.

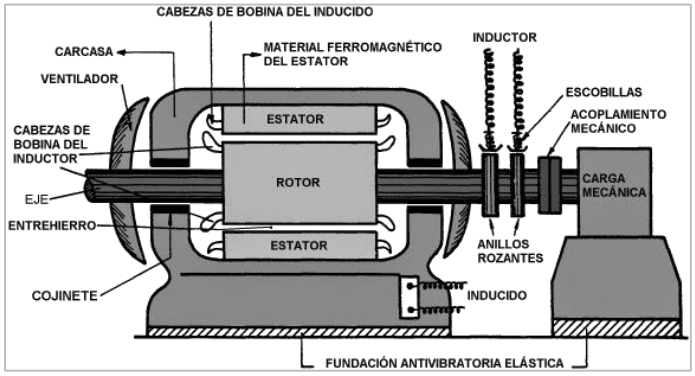


Figura 5. Partes principales de un generador síncrono [18].

Los polos magnéticos del rotor pueden ser tanto salientes como entrantes, un polo saliente es un polo magnético proyectado hacia afuera del eje del rotor y un polo entrante es un polo magnético construido al mismo nivel de la superficie del rotor. En la figura 6 se puede ver los tipos de rotor.

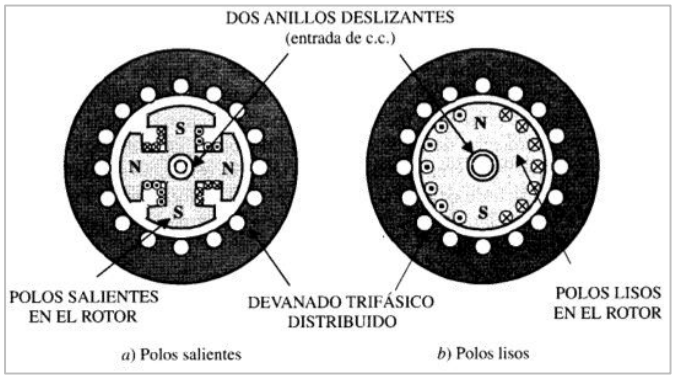


Figura 6. Tipos de polos [19].

Los anillos rozantes son anillos de metal que circundan por completo al eje de una máquina eléctrica, pero se encuentran aislados de ella. Un extremo del devanado del rotor de CD está unido a cada uno de los dos anillos rozantes del eje de la máquina síncrona y una escobilla estacionaria se desliza sobre cada anillo rozante.

Escobilla es un término que se refiere a un bloque hecho de un compuesto de carbón parecido al grafito que conduce electricidad libremente, pero tiene una fricción muy baja, por lo que no desgasta al anillo rozante.

Tipos de motores síncronos. Estos motores difieren de los descritos en la construcción del rotor, pero utilizan el mismo diseño de estator. Al igual que los motores de inducción se pueden construir con estatores monofásicos o trifásicos [17].

Motores CD sin escobillas. Se llaman motores CD sin escobillas puesto a que operan con una fuente de potencia de corriente directa, pero no tienen ni conmutadores ni escobillas. El rotor es similar al de un motor de avance paso a paso de imán permanente, excepto en que es de polos no salientes. El estator puede tener tres fases o más, lo que lo hace la mejor opción para ser usado como un generador eléctrico. Los componentes básicos de un motor de CD sin escobillas son:

1. Rotor de imán permanente.
2. Estator con un devanado de tres, cuatro o más fases.
3. Sensor de posición del rotor.
4. Circuito equivalente para controlar las fases del devanado del rotor.

Un motor de CD sin escobillas funciona por medio de la energización de una bobina del estator a la vez que con un voltaje de CD constante. Cuando se enciende una bobina, se produce un campo magnético del estator $\boldsymbol{B_S}$ y se produce un par en el rotor dado por la ecuación 1:

$$\tau_{ind} = k\boldsymbol{B_R} * \boldsymbol{B_S} \text{…ecuación 1}$$

Que tiende a alinear el rotor con el campo magnético del estator. En el momento que se muestra el campo magnético del estator $\boldsymbol{B_S}$ apunta hacia la izquierda mientras que el campo magnético del rotor de imán permanente $\boldsymbol{B_R}$ apunta hacia arriba, lo que produce un par en sentido contrario al de las manecillas del reloj en el rotor. Como resultado, el rotor girará hacia la izquierda. Donde k es una constante de par.

Si la bobina permanece energizada todo el tiempo, el rotor girará hasta que los dos campos magnéticos se alinearán y luego se detendría, al igual que un motor de avance paso a paso. La clave del funcionamiento de un motor de CD sin escobillas es que incluye un sensor de posición, por lo que el circuito de control sabe cuándo el rotor está casi alineado con el campo magnético del estator. En ese momento apagará la bobina A y prenderá la bobina B, lo que provocará que el rotor experimente una vez más un par en sentido contrario al de las manecillas del reloj y continúe girando. Este proceso se repite en forma indefinida y las bobinas se prenden en el siguiente orden: A, B, C, D, – A, – B, – C, – D, etc., por lo que el motor gira continuamente (figura 7).

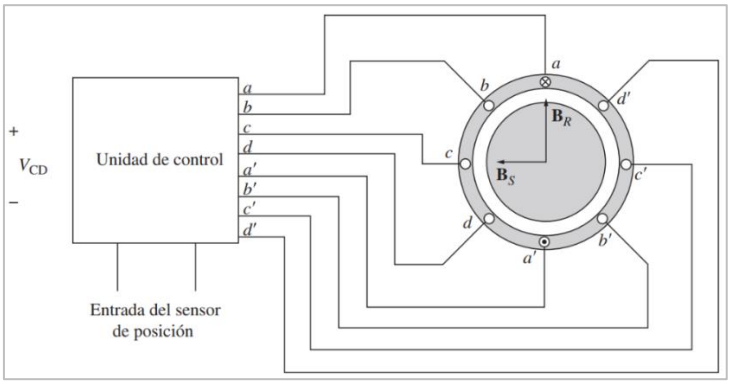


Figura 7. Representación de un motor sin escobillas [20].

Los motores de CD sin escobillas sólo están disponibles en tamaños pequeños, pero tienen muchas ventajas debido a la gama de tamaños con que se cuenta. Algunas de las ventajas más importantes son:

1. Eficiencia relativamente alta.
2. Larga vida útil y alta confiabilidad.
3. Poco o ningún mantenimiento.
4. Muy poco ruido de radiofrecuencia en comparación con los motores de cd con escobillas.
5. Se pueden lograr velocidades muy altas (mayores a 50,000 rpm) [17].

Debido a que por las terminales del motor sin escobillas se puede obtener una señal trifásica que presenta menos pérdidas durante su rectificación en comparación con una señal monofásica, tiene una buena curva de torque y potencia, cuenta con un peso reducido y no presenta componentes sometidos a fricción ya que no cuentan con escobillas por lo que prácticamente no necesita mantenimiento [21], es el elegido para usarse como generador en el presente proyecto.

a.2.5. Baterías

Batería se define como un grupo de dos o más celdas conectadas entre sí con el fin de suministrar corriente eléctrica. Celda es la unidad más elemental para convertir energía química en energía eléctrica. Una batería se ensambla combinando dos o más celdas [22].

El principio de funcionamiento de una batería está basado básicamente en un proceso reversible llamado reducción-oxidación, donde uno de los componentes pierde electrones, llamado oxidación y el otro componente gana electrones, llamado reducción. En este proceso los componentes no se consumen, únicamente cambian su estado de oxidación; por lo que dichos componentes pueden retornar a su estado original en las circunstancias adecuadas. Estas circunstancias son el cierre del circuito externo durante el proceso de descarga y la aplicación de una corriente externa durante el proceso de carga.

La capacidad de una celda es la cantidad total de electricidad producida en la reacción electroquímica, la cual se suele medir en miliamperios-hora (mAh). En la práctica, la capacidad de una batería se calcula descargando dicha celda a una intensidad de corriente determinada hasta alcanzar un valor especificado de tensión en sus bornes, denominada tensión de corte.

De esta forma, el valor de la capacidad es el producto de dicha intensidad de corriente de descarga y su duración de descarga en horas. Por este motivo, la capacidad de la batería se mide en miliamperios-hora (mAh) [23].

Batería de ion de litio: Su curva característica se puede observar en la figura 8.

- Tensión nominal de celda de 3.6 V a 3.8 V.
- Auto descarga 2% a 10% por mes.
- Densidad de energía 84 a 210 $\frac{Wh}{kg}$ y 168 a 555 $\frac{Wh}{dm^3}$.
- 3500 ciclos de vida.
- Daño de la batería al exponerla a altas temperaturas, crea un riesgo de incendio.

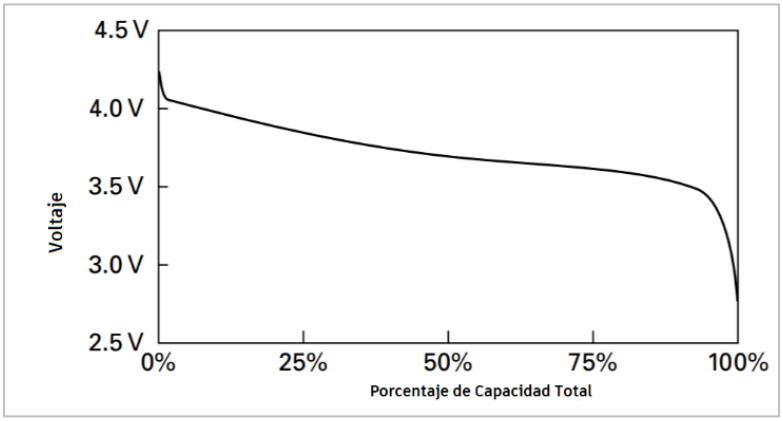


Figura 8. Descarga característica de batería de litio [24].

Hay una serie de consideraciones que influyen en la selección de una batería: energía o capacidad, potencia, número de ciclos, tamaños disponibles, requisitos de reciclaje y costo. La figura 9 compara las densidades de energía basadas en peso y volumen para los sistemas de baterías recargables más comúnmente disponibles. El litio es un claro ganador, pero tiende a ser el más caro [22].

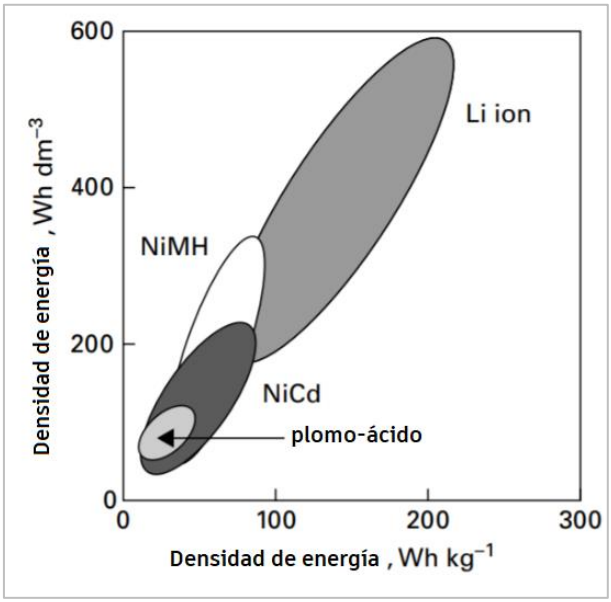


Figura 9. Comparación de densidad de energía por densidad y densidad de energía por peso de baterías secundarias [25].

Por lo tanto, las baterías de litio son una buena opción para este proyecto ya que tienen una relación de tamaño y peso además de que almacenan una mayor cantidad de energía.

b. Estado del arte

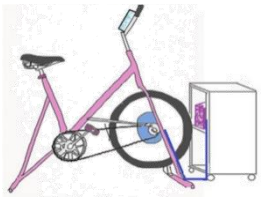
Existen ruedas generadoras de energía que transforman una bicicleta convencional en una eléctrica, estas ruedas son vendidas en forma de kit en tiendas en línea o convencionales y son de fácil instalación para el usuario.

Los elementos que las componen son la batería, que es donde se acumula y suministra la energía eléctrica usada para alimentar al controlador, que está encargado de mover y regular la cantidad de energía eléctrica suministrada por la batería hacia el motor, que tiene la función de asistir en el pedaleo, desconectándose cuando el ciclista no pedalea o frena, de tal manera que este no tenga que aplicar un esfuerzo extra para ir en contra del movimiento del motor; sensor de pedaleo, el cual detecta el pedaleo y envía una señal al controlador, a través de un sensor magnético, que ajusta el nivel de asistencia que debe proporcionar el motor y una pantalla de control de asistencia, que permita al usuario monitorear el nivel de torque proporcionado por el motor, el nivel de carga en la batería, la distancia recorrida, velocidad, etc. En algunas bicicletas los niveles se pueden monitorear a través de una aplicación móvil.

Las ruedas y bicicletas eléctricas comerciales tienen un sistema similar para la generación y almacenamiento de energía eléctrica [1].

A continuación, en la tabla 2 se presentan proyectos relacionados a ruedas y bicicletas eléctricas:

Ítem	Nombre	Descripción	Características	País	Instituto	Tipo	Año	
1	Sistema híbrido electromecánico para transporte y generación de energía [7].	Se desarrolla un modelo de bicicleta híbrida electromecánica que puede ser utilizada para transporte y generación de electricidad.	bicicleta rodada 26, motor eléctrico de 350 Watts, batería de litio LiNiMnCoO2.	México	IPN - ESIME	Tesis	2017	
2	Implementación de un sistema de carga y almacenamiento de energía eléctrica aprovechando la energía cinética de una bicicleta [8].	Se busca diseñar un sistema de generación de energía eléctrica para la carga de baterías, conformado por un generador eléctrico, un circuito regulador de voltaje y bancos de baterías, aprovechando la energía mecánica, en este caso una bicicleta, y posteriormente utilizar la energía almacenada en carga de dispositivos.	Bicicleta rodada 26, banco de baterías de 5 V a 1 A.	México	IPN – ESIME Zacatenco	Tesis	2018	
3	Análisis para la recuperación de energía cinética no aprovechada por medio del rin de una bicicleta [9].	Se diseña un sistema capaz de transformar la energía no aprovechada, generada por el pedaleo en una bicicleta estándar, que será captada a través de la llanta, que se acopla un motor de corriente directa, servirá como generador para suministrarla energía a una batería recargable.	Tensión a 12 V Corriente a 1 A	México	IPN – ESIME Azcapotzalco	Tesis	2017	
4	Diseño e implementación de un sistema de asistencia eléctrico en una bicicleta mediante la reutilización de baterías de Ni-MH [10].	Se realiza el modelado, diseño y construcción de un sistema de asistencia eléctrico aplicado a una bicicleta convencional, el cual el sistema es alimentado mediante baterías de Ni- MH recicladas y regeneradas.	Batería Ni-MH de 9.6 V, resistencia interna 8 mΩ. Rines de aluminio 26”.	Ecuador	Universidad Politécnica Salesiana Sede Cuenca	Proyecto técnico	2019	

Ítem	Nombre	Descripción	Características	País	Instituto	Tipo	Año	
5	Implementación de una máquina aeróbica para la inyección de energía al suministro eléctrico de CFE [11].	Se plantea suministrar energía eléctrica a CFE, por medio de una máquina aeróbica.	Tensión 127 V Frecuencia 59.2 – 60.8 Hz	México	IPN - UPIITA	Tesis	2015	
6	Rueda de Copenhagen [12]	Rueda trasera eléctrica de bicicleta que transforma una convencional en híbrida. Este tipo de rueda eléctrica ayuda al usuario en el pedaleo que realiza por medio de un motor interno y con múltiples sensores que se adecuan a las especificaciones del usuario y la vez controlarse por medio de un teléfono inteligente [44].	50 km de autonomía. Velocidad máxima de 25 km/h. Motor de 250 W. Batería Litio con vida de 1000 ciclos.	Dinamarca	Superpedestrian	Producto	2015	
7	Rueda Urban X Eco [13]	Rueda frontal de bicicleta que se convierte en eléctrica después de 60 segundos de iniciar el pedaleo [45].	Motor de 240 W. Velocidad Máxima de 14 km/h. Autonomía de 48 km.	Estados Unidos	UrbanX	Producto	2017	
8	Rueda GeoOrbital	Rueda frontal de bicicleta, que se transforma de una convencional a eléctrica. Utiliza un solo generador para la generación de energía.	Velocidad máxima de 32 km/h Motor de 36 V 500W Autonomía de hasta 20 km sin pedalear Autonomía de hasta 48 km pedalenado	Estados Unidos	GeoOrbitak	Producto	2015	

c. Planteamiento del problema

En la investigación realizada para este trabajo, los modelos de ruedas generadoras de energía eléctrica encontrados incluyen un solo generador para la generación y almacenamiento de energía en una batería no removible que ayuda al usuario en la tracción mecánica durante su trayecto. Antes de realizar el recorrido de la bicicleta, la batería de la rueda se debe cargar conectándose directamente a la toma de corriente y tarda en cargarse 3 horas o más [6].

El proyecto está financiado por la compañía GRUPO MONTES DE OCA Y DE LA HOZ S.A. de C.V., la empresa busca que la rueda pueda ser adaptada a los modelos de bicicletas mecánicas y eléctricas pertenecientes a los servicios de bicicletas compartidas existentes en el centro de la Ciudad de México, por lo que los autores se enfocarán en crear únicamente una rueda frontal y en un tamaño en específico, ya que el motor que impulsa las bicicletas eléctricas de los servicios de bicicletas compartidas se encuentra en la rueda trasera.

Los principales sistemas de bicicletas compartidas dentro de la Ciudad de México como ECOBICI, Mobike, Ofo y VBike no incluyen la funcionalidad de cargar baterías portables y su sistema de cobro se basa en el uso de un dispositivo móvil [3], por lo cual, se propone como trabajo terminal la construcción de un dispositivo incluido dentro de una rueda frontal de bicicleta que genere y almacene energía eléctrica en una batería removible, para que posteriormente el usuario pueda cargar su dispositivo móvil.

Aunado a esto, debido a que varios modelos de bicicletas eléctricas no pueden cargar la batería que alimenta al motor durante su trayecto, sino que se debe conectar directamente a la toma de corriente antes de usarla, se pretende añadir más de un generador a la rueda generadora de energía con la intención de poder alcanzar la tensión necesaria para alimentar esa batería.

Con el fin de mejorar la generación de energía eléctrica, los autores se plantean explorar la opción de construir una estructura situada dentro del rin de una rueda frontal de bicicleta de 26 pulgadas que incluya uno o más generadores eléctricos para aumentar la tensión entregada a la batería, posicionando el generador o generadores en puntos estratégicos dentro de la rueda para aumentar su rotación y mejorar su eficiencia en la generación de energía eléctrica, sin dañar su misma estructura.

Se incluirá dentro de la estructura un circuito que acondicione la señal o señales de cada generador con el propósito de obtener una salida final de 5V/2 A o 54.6V/2A.

Asimismo, si la rueda se integrara a un modelo de bicicleta eléctrica para cargar una batería portátil, el usuario podría recuperar una parte de la energía mecánica del motor que impulsa la bicicleta y ni siquiera tendría que pedalear para iniciar una generación de energía limpia durante todo su trayecto.

Se adquirirán el neumático, el rin, el buje de la rueda y los generadores eléctricos, es decir, no serán manufacturados por los autores del presente trabajo.

d. Propuesta solución

En la propuesta solución se presentan las metodologías de diseño a seguir y distintas consideraciones para llevar a cabo el desarrollo del proyecto.

d.1. Definición de la metodología

La metodología que se llevará a cabo está basada en el modelo V propuesto por Isermann [26] y se eligió debido a la relación que crea entre la fase de diseño y construcción mecatrónica, identificando las fases del proyecto que presentaron problemas y pudiendo realizar así las iteraciones necesarias para alcanzar los objetivos planteados.

d.2. Modelo V propuesto por Isermann

El modelo "V", donde la letra "V" significa Verificación y Validación (figura 10) muestra el avance de un proyecto mecatrónico de izquierda a derecha y cómo se relacionan las actividades de prueba con el análisis y diseño, sabiendo así a qué fase de desarrollo se debe volver si es que se encuentran fallas en alguna de las pruebas correspondientes, realizando las iteraciones pertinentes necesarias para optimizar cada aspecto del proyecto [26], por esta razón es que se eligió como la metodología de diseño a usarse.

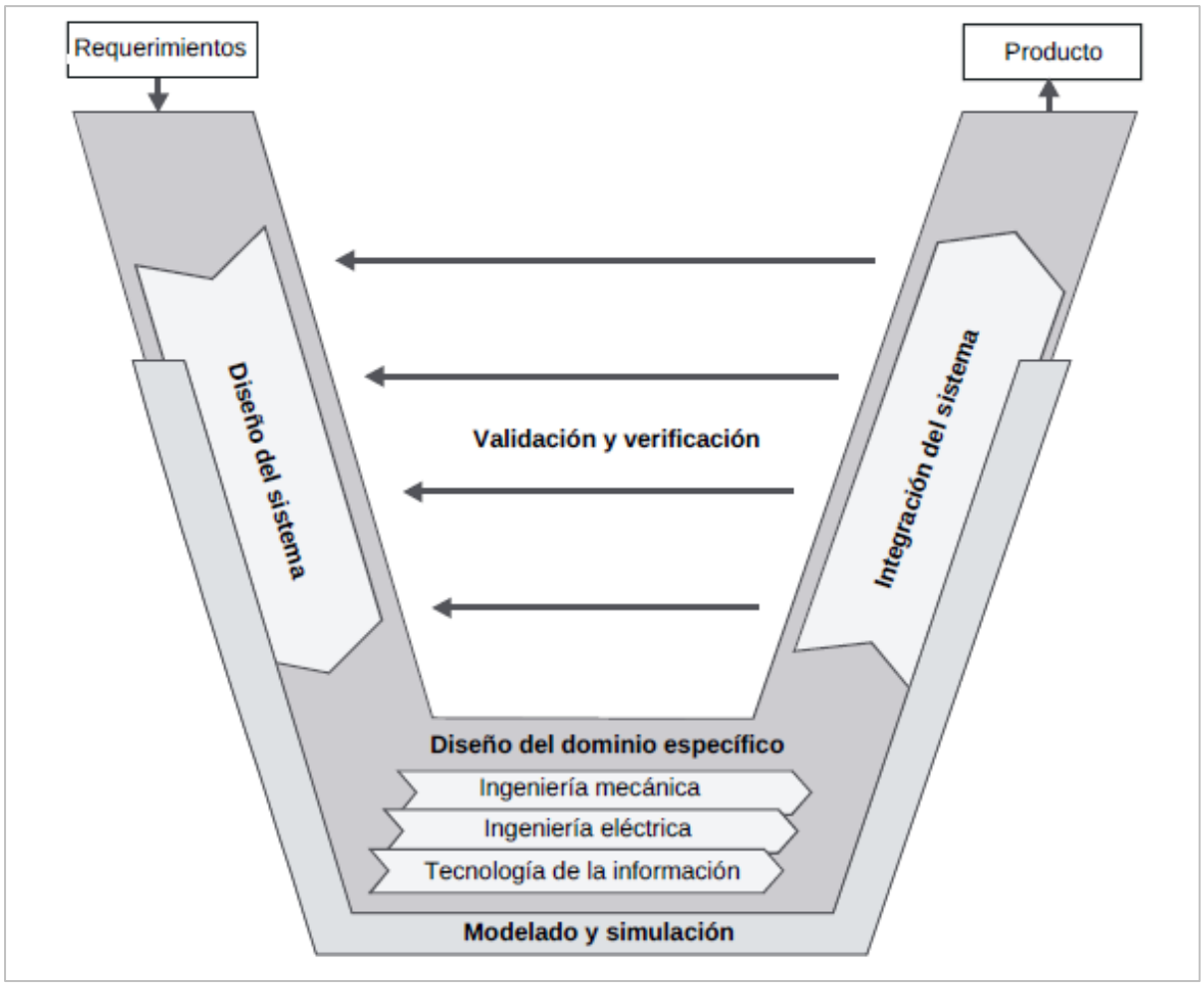


Figura 10. Modelo VDI 2206 para el diseño de sistemas mecatrónicos [26].

d.3. Desarrollo de la propuesta solución

Para el desarrollo del proyecto se analiza la posibilidad de reemplazar los radios convencionales de una rueda frontal de bicicleta tamaño 26 pulgadas por una estructura diseñada por los autores del presente trabajo, situada dentro del rin y sujetada a la horquilla.

Se integra uno o más generadores eléctricos al sistema, situándose fuera del eje de rotación de la rueda para aprovechar de mejor manera su giro y aumentar la tensión suministrada, con el fin de poder alimentar una batería portátil con una capacidad máxima de 5,000mA o una batería de bicicleta eléctrica con una capacidad máxima de 10,000mA.

Dentro de la estructura se incorpora una placa de circuito impreso que acondicione las señales creadas por los generadores, las conecte entre sí y por medio de un convertidor CC-CC aumente o reduzca la cantidad de tensión entregada a la batería, con el fin de otorgar 5V/2A o 54.6V/2A dependiendo de si el usuario quiere cargar una batería portable o la batería de la bicicleta eléctrica que está utilizando. El circuito impreso establecerá una comunicación Bluetooth con un dispositivo móvil para que a través de una aplicación desarrollada en el sistema operativo Android, se controle la elija la tensión entregada a la batería e indique el tiempo transcurrido y la ruta recorrida por el usuario (figura 11).

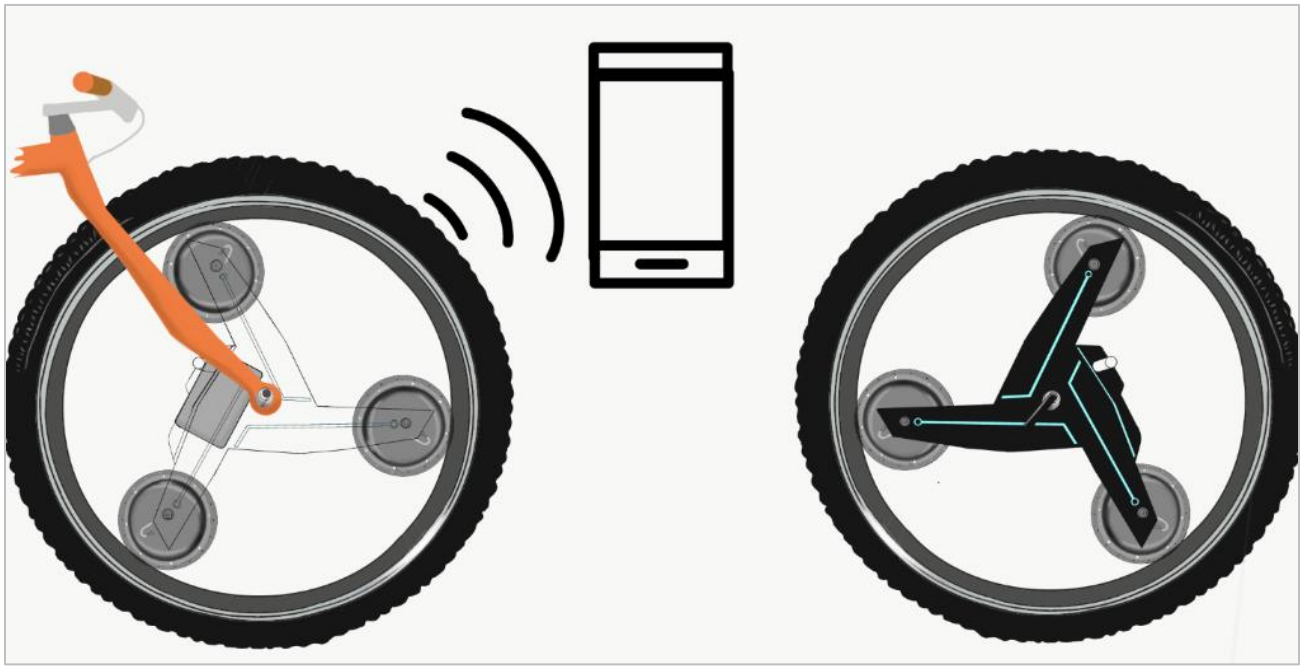


Figura 11. Bosquejo de la rueda frontal generadora de energía eléctrica.

De acuerdo con lo mencionado en [21], se utilizará un motor síncrono sin escobillas para el desarrollo del proyecto. Una posible solución temprana es utilizar un motor sin escobillas comercial que se adapte a bicicletas eléctricas.

Viendo las características de las dos configuraciones de motor sin escobillas, podemos llegar a la conclusión de que el motor de buje directo es una solución adecuada, ya que, por ser un motor directo, no contiene un reductor mecánico de velocidad que consuma parte de la energía del ciclista al pedalear.

Por lo tanto, se tomará como base el siguiente motor sin escobillas para bicicleta de la marca MXUS modelo XF07 (figura 12) y usando la herramienta de simulación de la empresa GRIN

TECHNOLOGIES [28] que se especializan en la comercialización de motores y bicicletas eléctricas, obtendremos su comportamiento cuando se utiliza como motor para tomarlo como referencia cuando se utilice como generador.

Características del motor sin escobillas para bicicleta MXUS XF07 [29]:

- Modelo: MXUS XF07
- Voltaje nominal (V):24V, 36V y 48V
- Potencia nominal (W):250 Watts
- Velocidad nominal (km/h):25-28 km/h
- Peso (Kg):3
- Polos magnéticos (2p):10
- Eficiencia (%): ≥ 78%
- Nivel de ruido (dB) <55

Figura 12. Motor sin escobillas MXUS XF07 para bicicleta [29].

Como se puede observar, el motor cuenta con una potencia nominal de 250 Watts, con tensión de 24 hasta 48V y una eficiencia mayor al 78%. En la figura 13 se muestra una simulación del dispositivo utilizado en modo motor, donde también se simula el controlador utilizado y el peso total del sistema, tomando en cuenta un peso de 20kg por parte de la bicicleta completa y considerando que el peso máximo del usuario sea de 100 kg. En la simulación realizada, la bicicleta se encuentra en un camino semiplano sin una pendiente abrupta.

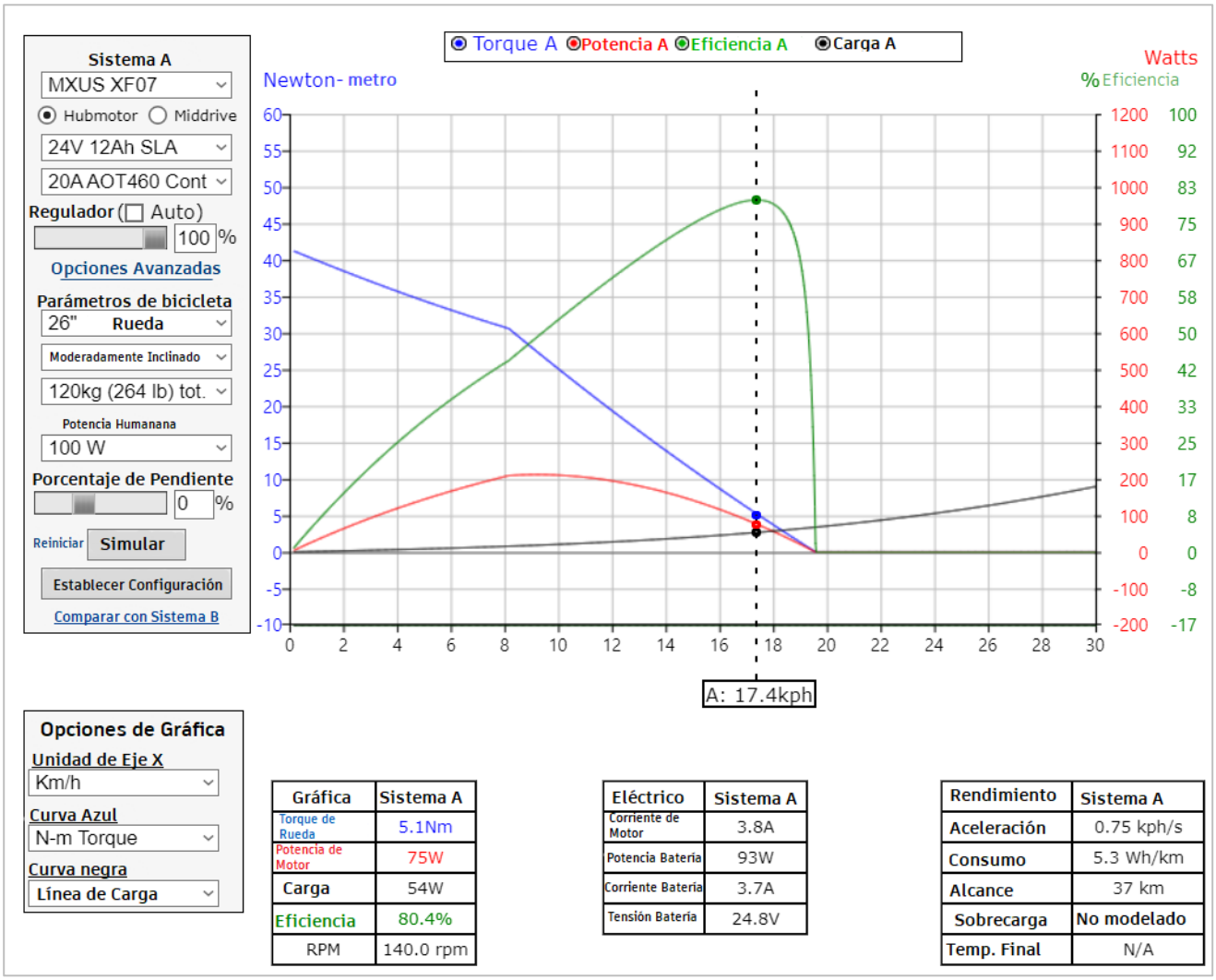


Figura 13. Simulación de motor XF07 con "Motor Simulator" de ebikes.ca. [28]

Los datos obtenidos son recopilados tomando en cuenta que el motor se encuentra trabajando a su mayor eficiencia que es de 80.4%, a una potencia en el motor de 75 W y una corriente en los devanados de 3.8 A, a una velocidad de 140 rpm.

Suponiendo que se tiene que cargar una batería de 5000 mAh con una señal de salida de 5V/2A, usando el motor XF07 como un generador, instalado en una bicicleta donde se pedalea obteniendo una velocidad en la rueda de 140 rpm, generando un aproximado de 3.8 A de corriente eléctrica y tomando en cuenta que se tiene que filtrar la salida del motor para poder alimentar una batería, lo cual se consumirá energía, el tiempo estimado de carga de la baria sería el siguiente, usando el valor de carga eléctrica de la batería que es de 5Ah.

$$T = \frac{5Ah}{2A} = 2.5\ horas$$…ecuación 2

El resultado es de 2.5 horas, pero tomando en cuenta una eficiencia del motor de 80% el tiempo de carga es el siguiente:

$$t = (2.5horas) * 0.2 + 2.5horas = 3horas$$…ecuación 3

La grafica de la simulación muestra que la corriente mínima es de 0.4 A, por lo cual, realizando el mismo cálculo anterior para una corriente de salida para alimentar la batería, se tendría que cargar en un tiempo total de:

$$t = (12.5horas) * 0.2 + 12.5horas = 15\ horas$$…ecuación 4

Por lo tanto, una batería con capacidad de 5000 mAh se estima que llegue a su carga completa en 3 horas a una velocidad promedio de 140 rpm del generador eléctrico.

Tomando en cuenta que una persona durante su trayecto no irá a una velocidad constante, se propone examinar la posibilidad de utilizar más de un generador eléctrico para garantizar que se entregue la mayor cantidad de tensión posible durante el tiempo de uso de la rueda para poder así alimentar la batería de una bicicleta eléctrica que recibe 54.6V/2A.

Debido a que se pretende utilizar el motor sin escobillas MXUS XF07 o un modelo parecido y por su tamaño este se puede adaptar a ruedas de 26 a 28 pulgadas, se elige una rueda de 26 pulgadas para utilizarse en este proyecto.

e. Justificación

Actualmente existen dispositivos parecidos a la propuesta planteada, pero el actual trabajo de investigación pretende estudiar los alcances de una rueda generadora de energía, ya que todos los ejemplares que se pueden observar en el mercado cuentan con un solo generador eléctrico y además este se encuentra situado en el eje de rotación de la rueda o de forma exterior, desperdiciando así energía mecánica que se podría aprovechar añadiendo uno o más generadores eléctricos dentro del rin de la rueda y colocándolos fuera del eje de rotación, aunado a esto, si las terminales del generador o los generadores eléctricos incluidos en la rueda se conectan en paralelo, se obtendría un aumento de tensión eléctrica utilizando convertidores CC-CC, por lo que se podría cargar una batería de bicicleta eléctrica durante el trayecto del usuario. Esto no existe actualmente, ya que las baterías de las bicicletas eléctricas encontradas en el mercado se deben cargar antes de usarse, conectándose a la toma de corriente.

No se encontraron ejemplares comerciales que realicen la carga de una batería portátil durante el trayecto de una bicicleta utilizando más de un generador para ello, la mayoría almacenan su energía en una batería no removible o se deben alimentar conectándolos a la toma de corriente, otra forma muy común de generación de energía eléctrica en bicicletas es haciendo uso de un dínamo, que se coloca externamente a la rueda.

Los autores del texto pretenden examinar los beneficios de rediseñar completamente la estructura de una rueda de bicicleta, incorporando dentro de ella uno o más generadores eléctricos comerciales que se puedan adaptar a su rin para aumentar la tensión entregada a una batería portátil sin dañarla o a la batería de una bicicleta eléctrica, por lo que el proyecto es una propuesta electromecánica innovadora.

El reto ingenieril del proyecto consiste en la aplicación de competencias mecánicas enfocadas a diseñar una estructura que logre un aumento en la rotación del generador o generadores eléctricos durante el giro de la rueda sin aumentar el arrastre de la misma ni aumente mucho su peso, permita integrar uno o más generadores en su interior y coloque correctamente el centro de masa de la rueda para que no se pierda su estabilidad, el empleo de competencias electrónicas al tratar la señal proveniente de los generadores e integrar un convertidor CC-CC para poder alimentar una batería portátil con una capacidad de máximo 5,000 mAh o una batería de bicicleta eléctrica con una capacidad de máximo 10,000 mAh, establecer una comunicación inalámbrica entre la rueda y el dispositivo móvil del usuario y el uso de competencias computacionales para el desarrollo de una aplicación móvil que controle el nivel de tensión entregada a la rueda, monitoree la distancia, tiempo, velocidad promedio, ruta y distancia recorrida por el usuario, todas ellas en conjunto integran las competencias profesionales mecatrónicas necesarias para el proyecto. De la rueda frontal generadora de energía eléctrica se debe obtener una salida de 5V/A o 54.6V/2A, utilizadas para alimentar una batería portátil o la batería de una bicicleta eléctrica sin dañarla, se podrá alimentar un tipo de batería a la vez, no las dos al mismo tiempo.

f. Objetivos

A continuación, se describe el objetivo general que debe satisfacer el proyecto y se presenta un desgolse de objetivos específicos que se deben seguir para alcanzar el objetivo general.

f.1. Objetivo general

Construir una rueda frontal de bicicleta de 26 pulgadas generadora de energía eléctrica con una salida de 5V/2A que alimente una batería portátil con capacidad máxima de 5000 mAh e incluir el desarrollo de una aplicación que indique por medio de un dispositivo móvil la cantidad de energía almacenada en una batería, la velocidad y distancia recorrida por el usuario.

f.2. Objetivos específicos del trabajo terminal 1

- Seleccionar el buje y rin de la rueda.
- Seleccionar los generadores de energía que se adapten al rin y buje de la rueda.
- Diseñar conceptualmente la estructura y los mecanismos de la rueda.
- Diseñar el acoplamiento mecánico entre los generadores y el rin de la rueda.
- Calcular las fuerzas externas que debe soportar la estructura de la rueda.
- Calcular los esfuerzos de la estructura.
- Determinar el material que cumpla los requerimientos mecánicos para la estructura de la rueda.
- Diseñar a detalle la estructura y los mecanismos de la rueda.
- Simular el comportamiento estático y dinámico del mecanismo integrado dentro de la rueda.

- Seleccionar las bibliotecas o APIs a usarse para llevar a cabo las funciones de la aplicación Android.
- Diseñar la interfaz de la aplicación Android.
- Seleccionar la batería portable y los componentes electrónicos para acondicionar la energía generada.
- Diseñar el sistema de acondicionamiento de energía eléctrica.
- Seleccionar los dispositivos electrónicos programables que lleven a cabo la comunicación inalámbrica entre la rueda y el dispositivo móvil.
- Establecer el código de control para los dispositivos electrónicos programables seleccionados.
- Diseñar el circuito impreso que integre los dispositivos electrónicos seleccionados.

f.3. Objetivos específicos del trabajo terminal 2

- Construir la estructura mecánica de la rueda.
- Incluir el rin, neumático y buje de la rueda.
- Incluir el sistema de generación y captación de energía en la estructura de la rueda.
- Construir el circuito impreso que integre todos los componentes electrónicos seleccionados.
- Incluir el circuito impreso en la estructura de la rueda.
- Verificar la energía generada usando el PCB ya integrado en la estructura de la rueda.
- Transferir la aplicación Android a un servidor remoto.
- Verificar el sistema de comunicación.
- Integrar la estructura de la rueda, el sistema de generación de energía, la placa impresa y la aplicación Android.
- Verificar el funcionamiento del sistema completo.
- Determinar la energía almacenada en la batería portátil durante un tiempo de 30 minutos.

Capítulo II. ANÁLISIS Y DISEÑO

a. División por áreas funcionales o disciplinas

La división por áreas funcionales del proyecto se realizó tomando en cuenta las habilidades de cada miembro del equipo.

a.1. Gráficas de Gant

Para la gráfica de Gant mostrada en las tablas 3 y 4 donde, D: Diego Cervantes Rodríguez, L: Luis Donaldo Arias García, V: Víctor Hisiquio Santiago, se muestra el cronograma de actividades seguido por los miembros del equipo para desarrollar el trabajo terminal de la rueda frontal de bicicleta generadora de energía eléctrica.

Tabla 3. Cronograma de actividades del trabajo terminal 1.

Cronograma de actividades Trabajo Terminal 1																
ID	Actividad	Responsable	Semana													
			1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	Clarificación de objetivos.	D, L, V														
1.1	Definición de funciones, módulos y sistemas.	D, L, V														
1.2	Definición de necesidades y requerimientos.	D, L, V														
2	Generación y evaluación de alternativas.	D, L, V														
4	Seleccionar el buje y rin de la rueda.	D, L, V														
5	Seleccionar el generador de energía que se adapte al rin y buje de la rueda.	D, L, V														
11	Seleccionar la batería portable.	D, L, V														
6	Estimación de costos.	V														
7	Calcular las fuerzas externas que debe soportar la estructura de la rueda.	D														
8.1	Generar modelo CAD de la bicicleta a donde se integrará la rueda.	D														
8.1	Diseño conceptual de la estructura mecánica dentro de la rueda.	D, V														
8	Diseñar el acoplamiento mecánico entre los generadores y el rin de la rueda.	D, V														
9.1	Diseñar el modelo CAD del sistema mecánico.	V														
8.2	Calcular los esfuerzos estáticos y dinámicos de la estructura.	D, V														
8.3	Determinar el material o materiales que cumplan los requerimientos mecánicos para la estructura de la rueda.	V														
9.2	Simular el comportamiento estático y dinámico del mecanismo integrado dentro de la rueda.	V														
10.1	Desarrollar el código que monitoree la velocidad, distancia y ruta del usuario en Android.	D														
10	Seleccionar las bibliotecas o APIs a usarse para llevar a cabo las funciones de la aplicación Android.	D														
10.2	Diseñar la interfaz de la aplicación Android.	D														
10.3	Verificar el funcionamiento de la aplicación Android.	D														
11.1	Seleccionar los dispositivos electrónicos programables que lleven a cabo la comunicación inalámbrica entre la rueda y el dispositivo móvil.	L, V														
12.1	Establecer el código de control para el dispositivo electrónico programable seleccionado.	L														
12	Diseñar el sistema de captación y acondicionamiento de energía eléctrica.	L,V														
12.2	Desarrollar el código que comunique la rueda con el dispositivo móvil por medio de Bluetooth.	L														
12.3	Diseñar y simular el circuito impreso que integre los dispositivos electrónicos seleccionados.	L														
13	Redacción del primer reporte a entregar de Trabajo Terminal 1.	D, L, V														
14	Redacción del segundo reporte a entregar de Trabajo Terminal 1.	D, L, V														
14	Simulación de la integración del sistema.	D, L, V														
15	Preparación para la defensa oral de proyecto.	D, L, V														
16	Redacción del reporte final a entregar de Trabajo Terminal 1.	D, L, V														

Tabla 4. Cronograma de actividades del trabajo terminal 2.

Cronograma de actividades Trabajo Terminal 2																				
ID	Actividad	Responsable	Semana																	
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	Construir la estructura de la rueda.	D, V																		
2	Integrar rin	D, V																		
2.1	Integrar neumático y cámara	D, V																		
2.2	Integrar buje de la rueda	D, V																		
3	Incluir el sistema de generación de energía en la estructura de la rueda.	D, V																		
5	Construir el circuito impreso que integre todos los componentes electrónicos seleccionados.	L																		
6	Verificar el funcionamiento del PCB construido.	L																		
4	Incluir y adaptar el sistema de captación de energía en la estructura de la rueda.	D, L, V																		
6	Incluir el circuito impreso en la estructura de la rueda.	D, L, V																		
7	Verificar la ganancia de energía generada usando el PCB ya integrado en la estructura de la rueda.	D, L, V																		
8	Transferir la aplicación Android a un servidor remoto.	D																		
	Veificar el funcionamiento de la aplicación Android.	D																		
9	Verificar el sistema de comunicación.	D, L																		
10	Integrar la estructura de la rueda, el sistema de generación de energía, la placa impresa y la aplicación Android.	D, L, V																		
11	Verificar el funcionamiento del sistema completo.	D, L, V																		

b. Diseño conceptual

Se presenta el diseño conceptual, en donde se desarrollan los requerimientos que debe cumplir el proyecto junto con los diagramas morfológicos pertinentes, en donde se hace la selección de alternativas.

b.1. Fijación de requerimientos

En la tabla 5 se presentan los requerimientos del sistema, describiendo si son demanadas (que describen requerimientos necesarios en el sistema) o deseables (que describen requerimientos prescindibles en el sistema).

Tabla 5. Requerimientos del sistema.

#	Requerimientos	Grado	
		Demandas	**Deseables**
1	**Desarrollo de la App**		
	Sistema operativo: Android.	X	
	API para utilizar en el monitoreo de la velocidad y distancia recorrida.	X	
2	**Costo de manufactura**		
	$20,000 MXN tentativamente.		X
3	**Conectividad**		
	Conectividad vía Bluetooth.	X	
	Comunicación inalámbrica de la estructura con un dispositivo móvil.	X	
4	**Trasmisión de datos**		
	Distancia máxima de trasmisión inalámbrica: 1metro.		X
5	**Dimensiones**		
	Tamaño de la rueda: 26 pulgadas.	X	
6	**Interfaz de la App**		
	Interfaz amigable y atractiva para el usuario en un dispositivo móvil.		X
7	**Energía**		
	Incluir uno o más generadores.	X	
	Alto rendimiento energético.		X

	Almacenar la energía eléctrica generada en una batería portátil.	X	
	Almacenar la energía eléctrica generada en una batería de bicicleta eléctrica.	X	
	Tensión y corriente de salida hacia la batería removible: 5V/2A.	X	
	Tensión y corriente de salida hacia la batería de la bicicleta eléctrica: 54.6V/2A.	X	
	Capacidad de carga de la batería portátil: 5,000 mAh.		X
	Capacidad de carga de la batería de bicicleta eléctrica: 10,000 mAh.		X
8	**Peso máximo del usuario**		
	Hasta 100 kg.	X	
9	**Monitoreo**		
	Monitorear y mostrar el tiempo del recorrido del usuario.		X
	Monitorear y mostrar la distancia recorrida por el usuario.	X	
	Monitorear y mostrar la velocidad del usuario.	X	
10	**Instalación/Mantenimiento**		
	Facilidad de instalación.		X
	Fácil mantenimiento y reparación.		X
	Modular y multiplataforma.	X	
11	**Compatibilidad**		
	Compatible con los servicios de bicicletas compartidas existentes en el centro de la Ciudad de México.		X

b.2. Definición de funciones, módulos y sistemas

Para realizar el diseño del sistema es necesario que se definan sus funciones y subfunciones, que se desempeñaran en el sistema de acuerdo con los requerimientos. Se utilizó la herramienta de modelo FBS (del inglés *Functional Breakdown Structure*), esta herramienta permite mostrar la importancia y las relaciones de diseño de las funciones, asociando la información de las tablas anteriores para definir el diseño del sistema.

Función principal del proyecto: Generar y almacenar energía eléctrica.

F1.- Transformar Energía Mecánica en Energía Eléctrica:

- El sistema debe ser capaz de transformar el movimiento mecánico cuando se mueva la bicicleta en energía eléctrica.

F2.- Acondicionar Señal:
- El sistema debe ser capaz de ajustar y rectificar la señal proveniente del generador para que posteriormente sea almacenada la energía eléctrica.
- El sistema debe limpiar la señal de ruido o interferencia no deseada.

F3.- Almacenar Energía Eléctrica:
- El sistema debe ser capaz de almacenar energía eléctrica en una batería portátil o en la batería de una bicicleta eléctrica, según el usuario lo defina.
- El sistema debe parar o seguir almacenando energía eléctrica a la batería dependiendo de la decisión del usuario.

F4.- Medir parámetros:
- El sistema debe ser capaz de trazar la ruta recorrida y medir el tiempo, la velocidad promedio y la distancia recorrida del usuario.

F5.- Gestionar Información:
- El sistema debe gestionar la tensión entregada a la batería dependiendo si es una batería removible o la batería de una bicicleta eléctrica.
- El sistema debe gestionar el envío de datos por medio de los protocolos usados con el módulo Bluetooth.

F6.- Comunicar los elementos electrónicos de la rueda con el dispositivo móvil:
- Transmitir información al dispositivo móvil del usuario.
- El sistema debe realizar una comunicación bluetooth con un dispositivo móvil.

F7.- Mostrar los parámetros medidos al usuario:
- En un dispositivo móvil se desplegará información acerca del tiempo, velocidad promedio, distancia recorrida y ruta del usuario.

F8.- Proteger, soportar e integrar todos los elementos de la rueda:
- La estructura interna de la rueda debe optimizar la rotación del generador, integrar uno o más generadores para aumentar la generación de energía y colocar el generador o generadores fuera del eje de rotación de la rueda.
- El sistema debe ser capaz de proteger todos los componentes dentro de la estructura, así como soportar el peso del usuario.
- Integrar los componentes del sistema.

A continuación, se muestra una descripción de las funciones en la tabla 6 y un desglose de éstas en la figura 14.

Tabla 6. Funciones del sistema.

F.- Generar y almacenar energía eléctrica	**F1.-** Transformar energía mecánica en energía eléctrica**.**	**F1.1.-** Transformar la energía mecánica para obtener energía eléctrica como salida.	
	F2.- Acondicionar la señal.	**F2.1.-** Ajustar la señal dependiendo de la tensión necesaria para la batería. **F2.2.-** Filtrar la señal de ruido.	
	F3.- Almacenar energía eléctrica.	**F3.1.-** Gestionar el almacenamiento de energía eléctrica.	
	F4.- Medir parámetros.	**F4.1.-** Medir los parámetros necesarios para saber el rendimiento de la ruta recorrida por el usuario.	
	F5.- Gestionar información.	**F5.1.-** Gestionar el inicio del recorrido. **F5.2.-** Gestionar los datos para el módulo de comunicación. **F5.3.-** Verificar si ha ocurrido un error en la transmisión de datos.	
	F6.- Transmitir información.	**F6.1.-** Comunicación Bluetooth entre el dispositivo móvil y la rueda.	
	F7.- Mostrar información.	**F7.1.-** Mostrar el tiempo transcurrido en el recorrido. **F7.2.-** Mostrar la velocidad promedio del usuario. **F7.3.-** Mostrar la distancia recorrida por el usuario.	

	F8.- Proteger, soportar e integrar.	**F8.1.-** Proteger los componentes del sistema. **F8.2.-** Soportar el peso del usuario. **F8.3.-** Integrar los componentes del sistema.	**F.8.3.1.-** Mejorar la eficiencia en la generación de energía eléctrica.

A continuación, se muestra el diagrama FBS (figura 14).

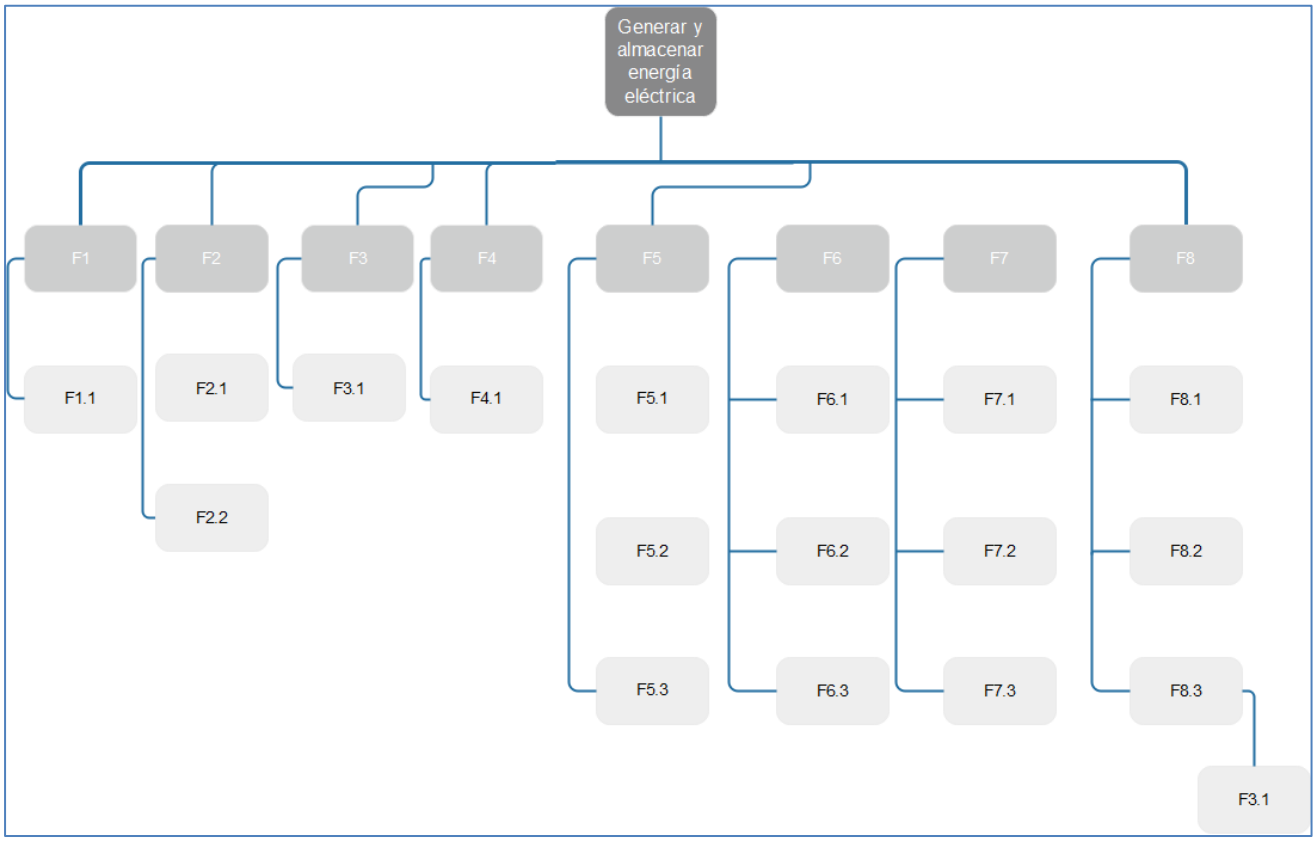


Figura 14. Diagrama FBS del sistema.

b.3. Generación de alternativas: Diagrama morfológico general

1. Lista de características o funciones esenciales de la rueda frontal de bicicleta, generadora de energía eléctrica:
    a. Generar energía eléctrica durante el trayecto de un ciclista.
    b. Panel de control.
    c. Acondicionar y almacenar la energía eléctrica generada.
    d. Medir, monitorear y mostrar datos de interés al usuario.

2. Medios por los que cada función podría realizarse:
    a. Generar energía eléctrica durante el trayecto de un ciclista.

i. 1 generador eléctrico colocado en el eje de rotación de la rueda frontal.
ii. 1 generador eléctrico colocado fuera del eje de rotación de la rueda frontal.
iii. Más de 1 generador eléctrico, colocados fuera del eje de rotación de la rueda frontal.
iv. 1 generador eléctrico colocado en el pedal de la bicicleta.
v. 1 generador eléctrico colocado en el eje de rotación de la rueda trasera.

b. Panel de control.

i. Control alámbrico, botones en el manubrio.
ii. Control inalámbrico, botones en el manubrio, módulo de radiofrecuencia integrado al circuito impreso de la rueda.
iii. Control inalámbrico, botones en el manubrio o dispositivo móvil, módulo Bluetooth integrado al circuito impreso de la rueda.
iv. Control inalámbrico, dispositivo móvil, microcontrolador que integre módulos Wi-Fi y/o Bluetooth.

c. Acondicionar y almacenar la energía eléctrica generada.

i. Sistema de generación, tratamiento y almacenamiento de energía dentro de una rueda de bicicleta.
ii. Sistema de generación, tratamiento y almacenamiento en el pedal de una bicicleta.
iii. Sistema de generación, tratamiento y almacenamiento de energía en toda la estructura de una bicicleta.

d. Medir, monitorear y mostrar datos de interés al usuario.

i. Leds en el manubrio.
ii. Dispositivo móvil del usuario.
iii. Leds en el manubrio y el dispositivo móvil del usuario.

3. Elaborar un diagrama que contenga todas las soluciones secundarias posibles después de haber analizado las distintas alternativas (tablas 7, 8, 9, 10, 11, 12 y 13).

Tabla 7. Diagrama morfológico general.

Función	Solución 1	Solución 2	Solución 3	Solución 4	Solución 5
Generar energía eléctrica durante el trayecto de un ciclista.	1 generador eléctrico colocado en el eje de rotación de la rueda frontal.	1 generador eléctrico colocado fuera del eje de rotación de la rueda frontal.	Más de 1 generador eléctrico, colocados fuera del eje de rotación de la rueda frontal.	1 generador eléctrico colocado en el pedal de la bicicleta.	1 generador eléctrico colocado en el eje de rotación de la rueda trasera.
Panel de control	Control alámbrico, botones en el manubrio.	Control inalámbrico, botones en el manubrio, módulo de radiofrecuencia integrado al circuito impreso de la rueda.	Control inalámbrico, botones en el manubrio o dispositivo móvil, módulo Bluetooth integrado al circuito impreso de la rueda.	Control inalámbrico, dispositivo móvil, microcontrolador que integre módulos Wi-Fi y/o Bluetooth.	X
Acondicionar y almacenar la	Sistema de tratamiento y almacenamiento de energía dentro de	Sistema de tratamiento y almacenamiento en	Sistema de tratamiento y almacenamiento de energía en toda la	X	X

energía eléctrica generada.	una rueda de bicicleta.	el pedal de una bicicleta.	estructura de una bicicleta.		
Medir, monitorear y mostrar datos de interés al usuario.	Leds en el manubrio.	Dispositivo móvil del usuario.	Leds en el manubrio y el dispositivo móvil del usuario.	X	X

b.3.1. Evaluación de alternativa 1: Diagrama morfológico general

Tabla 8. Alternativa general 1.

Función	Solución 1	Solución 2	Solución 3	Solución 4	Solución 5
Generar energía eléctrica durante el trayecto de un ciclista.	1 generador eléctrico colocado en el eje de rotación de la rueda frontal.	1 generador eléctrico colocado fuera del eje de rotación de la rueda frontal.	Más de 1 generador eléctrico, colocados fuera del eje de rotación de la rueda frontal.	1 generador eléctrico colocado en el pedal de la bicicleta.	1 generador eléctrico colocado en el eje de rotación de la rueda trasera.
Panel de control	Control alámbrico, botones en el manubrio.	Control inalámbrico, botones en el manubrio, módulo de radiofrecuencia integrado al circuito impreso de la rueda.	Control inalámbrico, botones en el manubrio o dispositivo móvil, módulo Bluetooth integrado al circuito impreso de la rueda.	Control inalámbrico, dispositivo móvil, microcontrolador que integre módulos Wi-Fi y/o Bluetooth.	X
Acondicionar y almacenar la energía eléctrica generada.	Sistema de tratamiento y almacenamiento de energía dentro de una rueda de bicicleta.	Sistema de tratamiento y almacenamiento en el pedal de la bicicleta.	Sistema de tratamiento y almacenamiento de energía en toda la estructura de una bicicleta.	X	X
Medir, monitorear y mostrar datos de interés al usuario.	Leds en el manubrio.	Dispositivo móvil del usuario.	Leds en el manubrio y el dispositivo móvil del usuario.	X	X

Alternativa 1:

Tabla 9. Resultado de la alternativa general 1.

Función	Generar energía eléctrica durante el trayecto de un ciclista.	Panel de control	Acondicionar y almacenar la energía eléctrica generada.	Medir, monitorear y mostrar datos de interés al usuario.
Alternativa 1	1 generador eléctrico colocado en el eje de rotación de la rueda frontal.	Control inalámbrico, botones en el manubrio, módulo de radiofrecuencia integrado al circuito impreso de la rueda.	Sistema de generación, tratamiento y almacenamiento en el pedal de una bicicleta.	Leds en el manubrio y el dispositivo móvil del usuario.

b.3.2. Evaluación de alternativa 2: Diagrama morfológico general

Tabla 10. Alternativa general 2.

Función	Solución 1	Solución 2	Solución 3	Solución 4	Solución 5
Generar energía eléctrica durante el trayecto de un ciclista.	1 generador eléctrico colocado en el eje de rotación de la rueda frontal.	1 generador eléctrico colocado fuera del eje de rotación de la rueda frontal.	Más de 1 generador eléctrico, colocados fuera del eje de rotación de la rueda frontal.	1 generador eléctrico colocado en el pedal de la bicicleta.	1 generador eléctrico colocado en el eje de rotación de la rueda trasera.
Panel de control	Control alámbrico, botones en el manubrio.	Control inalámbrico, botones en el manubrio, módulo de radiofrecuencia integrado al circuito impreso de la rueda.	Control inalámbrico, botones en el manubrio o dispositivo móvil, módulo Bluetooth integrado al circuito impreso de la rueda.	Control inalámbrico, dispositivo móvil, microcontrolador que integre módulos Wi-Fi y/o Bluetooth.	X
Acondicionar y almacenar la energía eléctrica generada.	Sistema de tratamiento y almacenamiento de energía dentro de una rueda de bicicleta.	Sistema de tratamiento y almacenamiento en el pedal de la bicicleta.	Sistema de tratamiento y almacenamiento de energía en toda la estructura de una bicicleta.	X	X
Medir, monitorear y mostrar datos de interés al usuario.	Leds en el manubrio.	Dispositivo móvil del usuario.	Leds en el manubrio y el dispositivo móvil del usuario.	X	X

Alternativa 2:

Tabla 11. Resultado de la alternativa general 2.

Función	Generar energía eléctrica durante el trayecto de un ciclista.	Panel de control	Acondicionar y almacenar la energía eléctrica generada.	Medir, monitorear y mostrar datos de interés al usuario.
Alternativa 2	1 generador eléctrico colocado en el eje de rotación de la rueda trasera.	Control alámbrico, botones en el manubrio.	Sistema de tratamiento y almacenamiento de energía en toda la estructura de una bicicleta.	Leds en el manubrio.

b.3.3. Evaluación de alternativa 3: Diagrama morfológico general

Tabla 12. Alternativa general 3.

Función	Solución 1	Solución 2	Solución 3	Solución 4	Solución 5
Generar energía eléctrica durante el trayecto de un ciclista.	1 generador eléctrico colocado en el eje de rotación de la rueda frontal.	1 generador eléctrico colocado fuera del eje de rotación de la rueda frontal.	Más de 1 generador eléctrico, colocados fuera del eje de	1 generador eléctrico colocado en el pedal de la bicicleta.	1 generador eléctrico colocado en el eje de rotación de la rueda trasera.

			rotación de la rueda frontal.		
Panel de control	Control alámbrico, botones en el manubrio.	Control inalámbrico, botones en el manubrio, módulo de radiofrecuencia integrado al circuito impreso de la rueda.	Control inalámbrico, botones en el manubrio o dispositivo móvil, módulo Bluetooth integrado al circuito impreso de la rueda.	Control inalámbrico, dispositivo móvil, microcontrolador que integre módulos Wi-Fi y/o Bluetooth.	X
Acondicionar y almacenar la energía eléctrica generada.	Sistema de tratamiento y almacenamiento de energía dentro de una rueda de bicicleta.	Sistema de tratamiento y almacenamiento en el pedal de la bicicleta.	Sistema de tratamiento y almacenamiento de energía en toda la estructura de una bicicleta.	X	X
Medir, monitorear y mostrar datos de interés al usuario.	Leds en el manubrio.	Dispositivo móvil del usuario.	Leds en el manubrio y el dispositivo móvil del usuario.	X	X

Alternativa 3:

Tabla 13. Resultado de la alternativa general 3.

Función	Generar energía eléctrica durante el trayecto de un ciclista.	Panel de control	Acondicionar y almacenar la energía eléctrica generada.	Medir, monitorear y mostrar datos de interés al usuario.
Alternativa 3	Más de 1 generador eléctrico, colocados fuera del eje de rotación de la rueda frontal.	Control inalámbrico, dispositivo móvil, microcontrolador que integre módulos Wi-Fi y/o Bluetooth.	Sistema de tratamiento y almacenamiento de energía dentro de una rueda de bicicleta.	Dispositivo móvil del usuario.

b.3.4. Método de objetivos ponderados: Diagrama morfológico general

Para seleccionar la mejor alternativa y soluciones secundarias.

1. Preparar la lista de objetivos de diseño:
    A. Generar y almacenar energía eléctrica durante el trayecto de un ciclista con el fin de almacenarla en una batería removible o en la batería utilizada para alimentar el motor de una bicicleta eléctrica.
    B. Mostrar datos de interés en una interfaz útil, amigable y atractiva para el usuario.
    C. Comunicación entre el usuario y el circuito impreso de la rueda para controlar la energía eléctrica entregada.
    D. Compatible con los servicios de bicicletas compartidas existentes en el centro de la Ciudad de México, modular y multiplataforma.

2. Ordenar la lista de objetivos (tabla 14):

Si A es más importante que B, se coloca 1, si no se coloca 0 y si son igualmente importantes se coloca $\frac{1}{2}$ en las dos posiciones correspondientes.

Tabla 14. Ordenar la lista de objetivos.

Objetivos	A	B	C	D	Totales de la fila
A	-	1	1	1	3
B	0	-	0	0	0
C	0	1	-	1	2
D	0	1	0	-	1

Los totales de las filas indican el orden de clasificación de objetivos resultante, el mayor indica que el objetivo de la fila es el más importante (tabla 15).

Tabla 15. Lista de objetivos ordenados.

Objetivos	Totales de la fila
A	**3**
C	**2**
D	**1**
B	**0**

3. Asignar a los objetivos ponderaciones relativas:

El número más alto corresponderá al 10 y respecto a esto, se irán ponderando las demás posiciones, dándole un valor de 1 al número más bajo (tabla 16).

Ejemplo:

3 – 10

2 – X

$X = \frac{2(10)}{3} = 6.6666 \approx 7$…ecuación 5

Tabla 16. Lista de objetivos con pesos ponderados.

Objetivos	Pesos relativos	Totales de la fila
A	**10**	**3**
	9	
	8	
C	**7**	**2**
	6	
	5	
	4	
D	**3**	**1**
	2	
B	**1**	**0**

A. Generar y almacenar energía eléctrica durante el trayecto de un ciclista con el fin de almacenarla en una batería removible o en la batería utilizada para alimentar el motor de una bicicleta eléctrica.

    a. Peso: 10

B. Mostrar datos de interés en una interfaz útil, amigable y atractiva para el usuario.

    a. Peso: 1

C. Comunicación entre el usuario y el circuito impreso de la rueda para controlar la energía eléctrica entregada.

    a. Peso: 7

D. Compatible con los servicios de bicicletas compartidas existentes en el centro de la Ciudad de México, modular y multiplataforma.

    a. Peso: 3

4. Establecer parámetros de rendimiento de los objetivos enlistados:

Escala cuantificable de 7 puntos, objetivo A (tabla 17):

Tabla 17. Escala de 7 puntos, calificación del parámetro del objetivo A.

Calificación del parámetro: Corriente y tensión eléctrica máxima de salida.	**Magnitud del parámetro a calificar: Corriente y tensión eléctrica máxima de salida.**	**A. Generar y almacenar energía eléctrica durante el trayecto de un ciclista con el fin de almacenarla en una batería removible o en la batería utilizada para alimentar el motor de una bicicleta eléctrica.**
0	Salida máxima < 1V/1A	Muy poco
1	Salida máxima < 2.5V/1A	Poco
2	Salida máxima < 4V/1.5A	Por debajo del promedio
3	Salida máxima = 5V/1A	Promedio
4	Salida máxima > 5V/2A	Por arriba del promedio
5	Salida máxima > 24V/2A	Bueno
6	Salida máxima <= 48V/10A	Extremadamente bueno

Escala cuantificable de 7 puntos, objetivo B (tabla 18):

Tabla 18. Escala de 7 puntos, calificación del parámetro del objetivo B.

Calificación del parámetro: Interfaz.	**Magnitud del parámetro a calificar: Interfaz.**	**B. Mostrar datos de interés en una interfaz útil, amigable y atractiva para el usuario.**
0	**No muestra ninguna de las siguientes:** 1.- Velocidad del usuario. 2.- Ruta recorrida por el usuario. 3.- Control para la energía eléctrica de salida.	Muy poco
1	**Muestra 1 de las siguientes:** 1.- Velocidad del usuario. 2.- Ruta recorrida por el usuario. 3.- Control para la energía eléctrica de salida.	Poco
2	**Muestra 2 de las siguientes:** 1.- Velocidad del usuario. 2.- Ruta recorrida por el usuario. 3.- Control para la energía eléctrica de salida.	Por debajo del promedio
3	**Muestra 2 de las siguientes en un dispositivo y 1 en otro:**	Promedio

	1.- Velocidad del usuario. 2.- Ruta recorrida por el usuario. 3.- Control para la energía eléctrica de salida.	
4	**Muestra 2 de las siguientes en un dispositivo y 2 en otro:** 1.- Velocidad del usuario. 2.- Ruta recorrida por el usuario. 3.- Control para la energía eléctrica de salida.	Por arriba del promedio
5	**Muestra 3 de las siguientes en un dispositivo y 1 en otro:** 1.- Velocidad del usuario. 2.- Ruta recorrida por el usuario. 3.- Control para la energía eléctrica de salida.	Bueno
6	**Muestra todas las siguientes en un mismo dispositivo:** 1.- Velocidad del usuario. 2.- Ruta recorrida por el usuario. 3.- Control para la energía eléctrica de salida.	Extremadamente bueno

Escala cuantificable de 7 puntos, objetivo C (tabla 19):

Tabla 19. Escala de 7 puntos, calificación del parámetro del objetivo C.

Calificación del parámetro: Comodidad e invasividad del control.	**Magnitud del parámetro a calificar: Comodidad e invasividad del control.**	**C. Comunicación entre el usuario y el circuito impreso de la rueda para controlar la energía eléctrica entregada.**
0	Incómodo e invasivo	Muy poco
1	Invasivo	Poco
2	Incómodo	Por debajo del promedio
3	Cómodo	Promedio
4	Muy cómodo	Por arriba del promedio
5	No invasivo	Bueno
6	Muy cómodo y no invasivo	Extremadamente bueno

Escala cuantificable de 7 puntos, objetivo D (tabla 20):

Tabla 20. Escala de 7 puntos, calificación del parámetro del objetivo D.

Calificación del parámetro: Instalación en distintas bicicletas mecánicas y eléctricas.	Magnitud del parámetro a calificar: Instalación en distintas bicicletas mecánicas y eléctricas.	D. Compatible con los servicios de bicicletas compartidas existentes en el centro de la Ciudad de México, modular y multiplataforma.
0	Afecta la estructura y el funcionamiento de la bicicleta	Muy poco
1	Afecta el funcionamiento de la bicicleta	Poco
2	Afecta la estructura de la bicicleta	Por debajo del promedio
3	-	Promedio
4	No afecta la estructura de la bicicleta	Por arriba del promedio
5	No afecta el funcionamiento de la bicicleta	Bueno
6	No afecta la estructura ni el funcionamiento de la bicicleta	Extremadamente bueno

5. Calcular y comparar los valores de unidad relativa de los diseños alternativos (tabla 21).

Tabla 21. Comparación de alternativas VS objetivos.

Función	Generar energía eléctrica durante el trayecto de un ciclista.	Medir, monitorear y mostrar datos de interés al usuario.	Panel de control	Acondicionar y almacenar la energía eléctrica generada.
Alternativa 1	1 generador eléctrico colocado en el eje de rotación de la rueda frontal.	Leds en el manubrio y el dispositivo móvil del usuario.	Control inalámbrico, botones en el manubrio, módulo de radiofrecuencia integrado al circuito impreso de la rueda.	Sistema de generación, tratamiento y almacenamiento en el pedal de la bicicleta.
Alternativa 2	1 generador eléctrico colocado en el eje de rotación de la rueda trasera.	Leds en el manubrio.	Control alámbrico, botones en el manubrio.	Sistema de tratamiento y almacenamiento de energía en toda la estructura de una bicicleta.

Alternativa 3	Más de 1 generador eléctrico, colocados fuera del eje de rotación de la rueda frontal.	Dispositivo móvil del usuario.	Control inalámbrico, dispositivo móvil, microcontrolador que integre módulos Wi-Fi y/o Bluetooth.	Sistema de tratamiento y almacenamiento de energía dentro de una rueda de bicicleta.

Considerar cada alternativa y calcular para cada una de ellas la calificación de su rendimiento. La multiplicación del peso de los diferentes objetivos por la calificación del parámetro da como resultado el valor general de utilidad (tabla 22).

Consideraciones: El motor seleccionado entrega una corriente en los devanados de 3.8 A, a una velocidad de 140 revoluciones por minuto.

Tabla 22. Calcular y comparar los valores de utilidad relativa de los diseños alternativos.

Objetivo	**Peso**	**Parámetro**	**Alternativa 1**			**Alternativa 2**			**Alternativa 3**		
			Magnitud	Calificación	Valor	Magnitud	Calificación	Valor	Magnitud	Calificación	Valor
A	**10**	Corriente y tensión eléctrica máxima de salida.	24-48V /3.8 A	**5**	**50**	24-48V /3.8 A	**5**	**50**	24-48V / N*3.8 A N = #de motores conectados en paralelo	**6**	**60**
B	**1**	Interfaz.	Por arriba del promedio	**4**	**4**	Por debajo del promedio	**2**	**2**	Extremadamente bueno	**6**	**6**
C	**7**	Comodidad e invasividad del control.	Por arriba del promedio	**4**	**28**	Poco	**1**	**7**	Bueno	**5**	**35**
D	**3**	Instalación en distintas bicicletas mecánicas y eléctricas.	Poco	**1**	**3**	Muy poco	**0**	**0**	Extremadamente bueno	**6**	**18**
Valor general de utilidad					**85**			**59**			**119**

Por lo tanto, podemos concluir que la mejor opción a tomar para desarrollar el proyecto de rueda generadora de energía eléctrica es la Alternativa 3 que tiene un valor general de utilidad de 119.

A continuación, se realiza el mismo procedimiento para crear el diagrama morfológico de los componentes necesarios para empezar a diseñar el proyecto.

b.4. Diagrama morfológico del generador de energía que se adapte al rin y buje de la rueda

Para la selección de motor se seleccionarán los motores que se adapten a las medidas del rin y su eficiencia sea mayor al 75%. Además de eso se tomaron en cuenta como objetivos a alcanzar, su voltaje nominal, potencia, precio y el peso del motor.

Los motores con características que son adecuados para este proyecto son los descritos en la tabla 23:

Tabla 23. Comparación de alternativas de motores VS objetivos.

	A	B	C	D	E	F	G	H	I	J	K	L	M
Voltaje Nominal [V]	36	48	48	36	48	36	48	36	36	48	24	24	48
Potencia [W]	250	500	1000	250	500	250	500	350	250	1000	250	350	350
Peso [Kg]	2.8	6	6.4	2.8	3.7	3.6	3.6	3.3	3	6	3	3.2	3.2
Costo [$]	3800	7234	7873	6595	5217	5217	5957	2619	3431	7034	2423	2729	2730

Se asigna una calificación a los objetivos más importantes (tabla 24):

Si A es más importante que B, se coloca 1, si no se coloca 0 y si son igualmente importantes se coloca 0.5 en las dos posiciones correspondientes.

Tabla 24. Ordenar la lista de objetivos del motor.

Objetivos	Voltaje nominal	Potencia	Precio	Peso (masa)	Calificación Total
Voltaje nominal	-	0.5	1	1	2.5
Potencia	0.5	-	1	1	2.5
Costo	0	0	-	0	0
Peso (masa)	0	0	1	-	1

Por la calificación total de la tabla anterior se va a ordenar la tabla para realizar el método de objetivos ponderados.

Los objetivos se ordenarán de la siguiente manera con sus respectivos pesos ponderados (tabla 25):

Tabla 25. Pesos ponderados de la lista de objetivos del motor.

Objetivos	Meta por alcanzar	Peso ponderado
Voltaje nominal	Alto voltaje nominal	0.3
Potencia	Alta potencia	0.3
Peso	Menor peso	0.25
Costo	Menor costo	0.15

La suma del peso ponderado de todos los objetivos debe dar en total un valor de uno.
En la tabla 27 se presentan las calificaciones según los objetivos de cada uno de los motores siguiendo una escala de once puntos (tabla 26), también se muestra los pesos que se les dieron a los objetivos según su nivel de importancia.

Tabla 26. Escala de 11 puntos.

Escala de once puntos	Significado
0	Solución totalmente inútil
1	Solución inadecuada
2	Solución muy mala
3	Solución mala
4	Solución tolerable
5	Solución adecuada
6	Solución satisfactoria
7	Solución buena
8	Solución muy buena
9	Solución excelente
10	Solución perfecta o ideal

Tabla 27. Calificación de los motores por cada objetivo siguiendo una escala de once puntos.

Objetivo	Peso	A	B	C	D	E	F	G	H	I	J	K	L	M
Voltaje	0.3	5	6	6	5	6	5	6	5	5	6	4	4	6
Potencia	0.3	4	6	7	4	6	4	6	5	4	7	4	5	5
Peso [kg]	0.25	6	3	3	6	4	5	5	5	6	3	6	5	6
Costo	0.15	4	3	3	3	3	3	3	5	4	3	5	5	5

Cada calificación se multiplico por el peso que se le dio a cada objetivo y se sumó en su correspondiente columna para obtener el valor general de utilidad, como se muestra en la tabla 28.

Tabla 28. Valor general de utilidad calculado según el motor.

Motor	A	B	C	D	E	F	G	H	I	J	K	L	M
Valor general de utilidad	4.8	4.8	5.1	4.65	5.05	4.4	5.3	5	4.8	5.1	4.65	4.7	5.55

Como se muestra en la tabla anterior, el motor con el mayor valor es el **M** y el que le sigue el motor **G**, las características de ambos se muestran enseguida (figura 15).

MXUS-XF07:	**Motor G:**
Voltaje nominal: 48V. Potencia nominal: 350 Watts. Peso: 3Kg. Modelo: XF07. Marca: XMUS. Posición: Rueda frontal. Polos magnéticos (2p): 10. Precio: $2730. Eficiencia: ≥ 78%. Grado de ruido (DB): <55. Grado IP: IP65.	Voltaje: 48V. Potencia nominal: 500 Watts. Peso: 3.6 Kg. Marca: Bafang / 8fun. Posición: Rueda frontal. Diseño: Sin escobillas. Precio: $ 5957. Eficiencia: ≥ 80%. Grado de ruido (DB): <55. Grado IP: IP65.

Figura 15. Motores MXUS XF07 y Bafang/8fun.

Como podemos observar de la tabla anterior, el motor **G** tiene una mejor potencia nominal que el motor **M**, pero el costo es mucho mayor en el motor **G**, por lo cual se toma en consideración un menor costo del material, llegando a la conclusión de que el motor **M** es el adecuado para este proyecto.

b.5. Seleccionar la rueda frontal

Para la selección del rin, neumático y cámara de la rueda frontal se debe tomar en cuenta que las dimensiones del motor elegido (figura 16) sean compatibles con el cuerpo del rin para que éstos hagan contacto, permitiendo la rotación del generador eléctrico al usar la rueda.

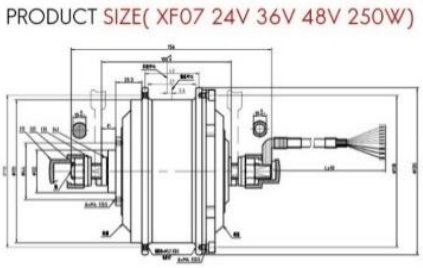


Figura 16. Dimensiones motor MXUS XF07.

Las dimensiones importantes del motor son de 100 mm aproximadamente de anchura total y 37 mm de ancho interior, en donde se pretende que haga contacto motor con el rin (figura 17).

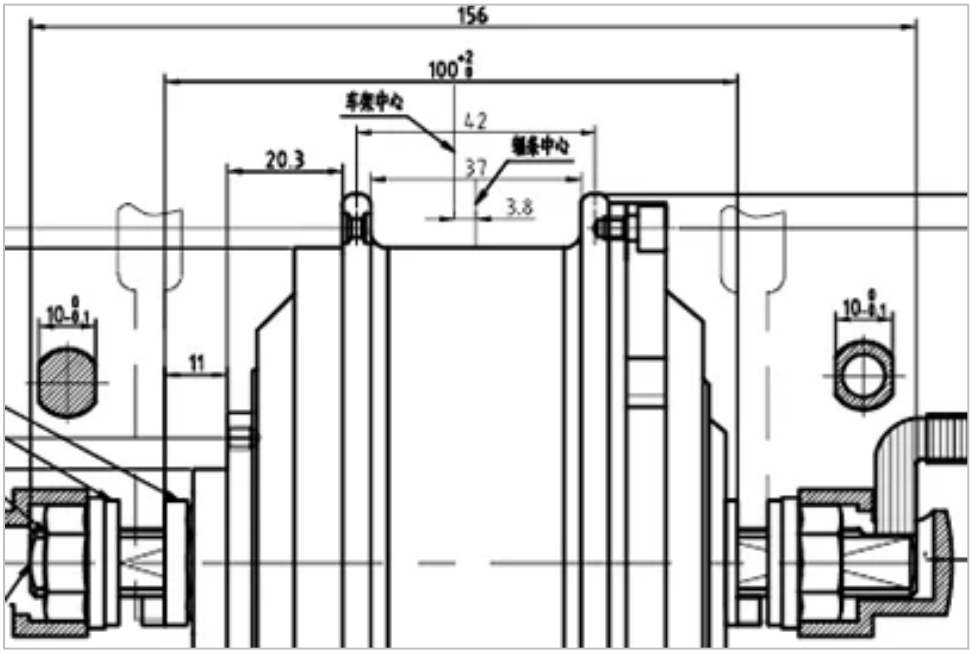


Figura 17. Dimensiones del motor MXUS XF07 importantes para la selección del rin.

Al adquirir la rueda frontal existen 2 opciones:

1) Obtener una rueda frontal y remover sus radios.
2) Conseguir el rin, cámara, neumático, buje y seguro del buje por separado para ensamblarlos.

Se considerarán las 2 opciones para elegir la más conveniente.

Antes de empezar a desarrollar el diagrama morfológico del rin, se debe considerar los tipos de rines que existen y encontrar el que más nos sirve.

- **Rin tipo Westwood:** Lo incorporan las clásicas bicicletas con frenos de varilla. No es adecuado para frenos de arco habituales, tan solo para los que actúan sobre el plano de la llanta (figura 18) [30].

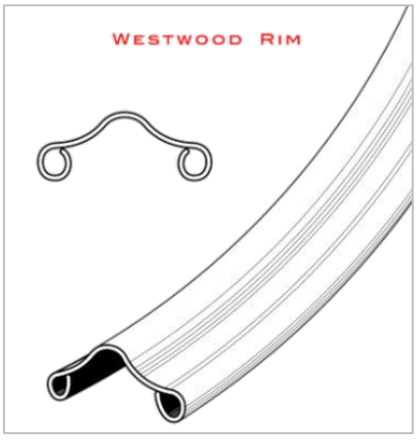


Figura 18. Rin tipo Westood.

- **Rin tipo Sprint:** Generalmente usados en bicicletas de pista (figura 19) [31].

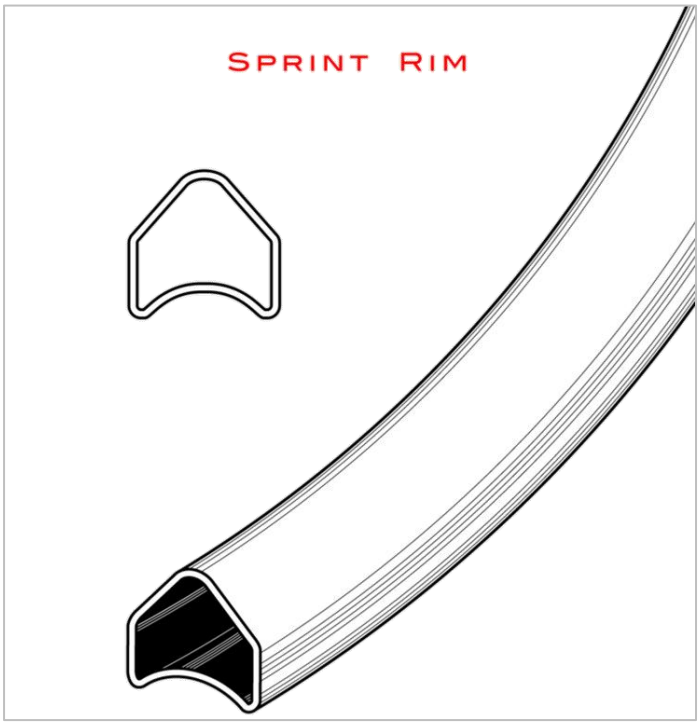


Figura 19. Rin tipo Sprint.

- **Llanta o rin tipo Endrick:** Instaladas en bicicletas deportivas de los años 1930 hasta los 1950, precursor de las modernas llantas con borde para freno de hoy en día (figura 20) [32].

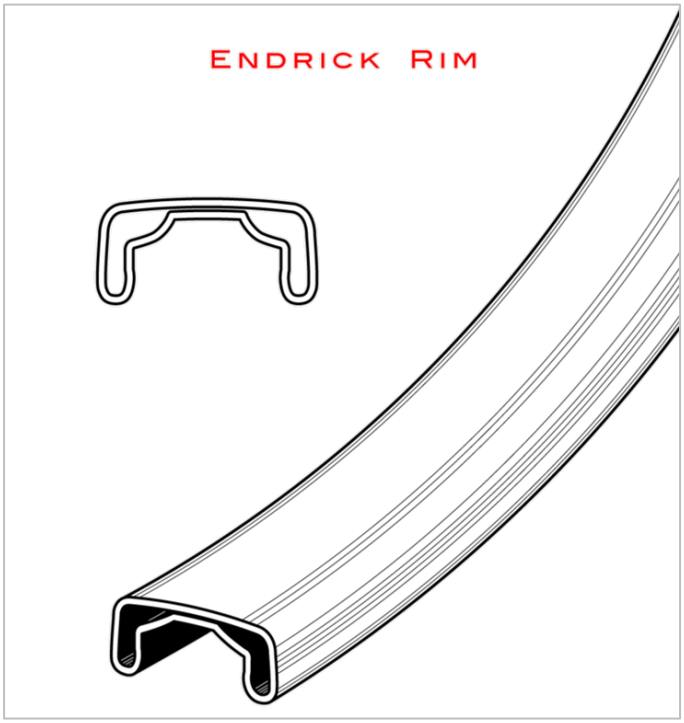


Figura 20. Rin tipo Endrick.

También se debe considerar la simbología de tamaños estandarizada respecto a la ISO, Marcas en pulgadas y en Código Francés, el tamaño del neumático descrito por esta simbología debe coincidir con el tamaño del rin (figura 21) [33].

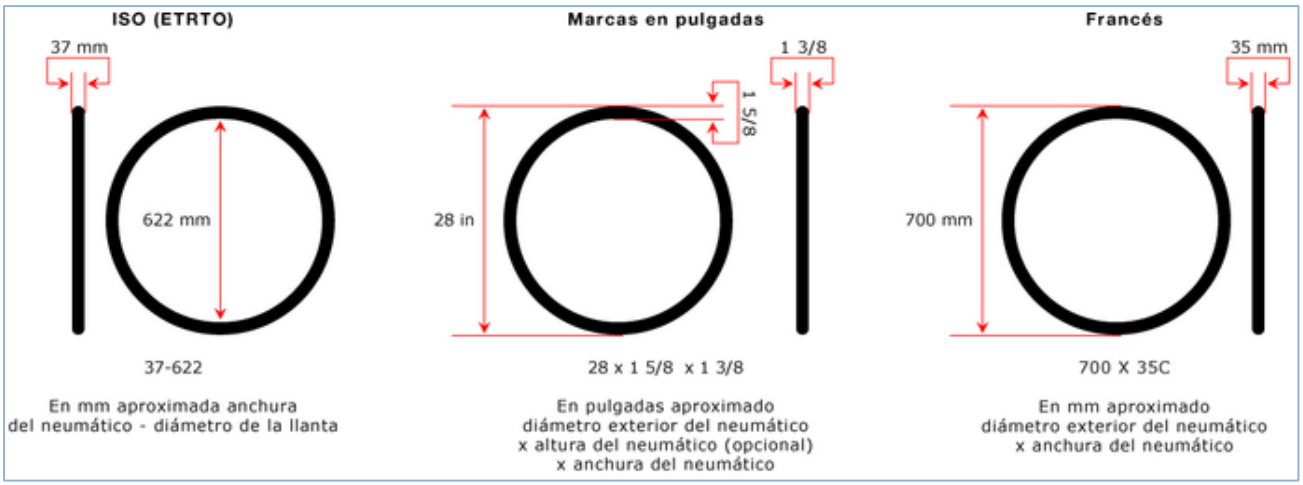


Figura 21. Simbología de tamaños de llanta.

Las opciones con características adecuadas para este proyecto son (tabla 29):

Tipo de rin:
W = Westwood …ecuación 6
S = Sprint …ecuación 7
E = Endrick …ecuación 8

Tamaño:
R24 = Rodada 24 …ecuación 9
R26 = Rodada 26 …ecuación 10
R28 = Rodada 28 …ecuación 11

Ancho interior, compatibilidad con motor:

- W = Westwood: …ecuación 12
    a. Ancho interior < 37 mm, Compatibilidad con motor = NO
    b. Ancho interior < 37 mm, Compatibilidad con motor = NO
    c. Ancho interior < 37 mm, Compatibilidad con motor = NO

- S = Sprint: …ecuación 13
    a. Ancho interior = Westwood, Compatibilidad = NO.

- E = Endrick: …ecuación 14
    a. Ancho interior = Westwood, Compatibilidad = NO.

Tabla 29. Comparación de alternativas de ruedas, rines, cámaras, neumáticos, bujes y seguros del buje.

	A	B	C	D	E
Tipo del rin	E	E	E	S	W
Tamaño	R26	R26	R26	R26 650B	R26
Ancho interior del rin [mm]	30	10	10	10	10
Precio de la rueda [$]	0	0	0	0	0
Precio rin [$]	0	0	0	790	225
Precio cámara [$]	0	0	0	79	79
Precio neumático [$]	0	0	0	695	695
Precio buje [$]	0	0	0	704	648
Precio del seguro del buje [$]	0	0	0	0	0
Precio total [$]	1049	1190	1190	2268	1647

Asignamos una calificación a los objetivos más importantes (tabla 30):

Si A es más importante que B, se coloca 1, si no se coloca 0 y si son igualmente importantes se coloca 0.5 en las dos posiciones correspondientes.

Tabla 30. Ordenar la lista de objetivos de las ruedas, rines, cámaras, neumáticos, bujes y seguros del buje.

Objetivos	Tipo del rin	Tamaño	Ancho interior del rin [mm]	Precio de la rueda [$]	Precio rin [$]	Precio cámara [$]	Precio neumático [$]	Precio buje [$]	Precio del seguro del buje [$]	Precio total [$]	Calificación total
Tipo del rin	-	0	0	1	1	1	1	1	1	1	7
Tamaño	1	-	0	1	1	1	1	1	1	1	8
Ancho interior del rin [mm]	1	1	-	1	1	1	1	1	1	1	9
Precio de la rueda [$]	0	0	0	-	0.5	0.5	0.5	0.5	0.5	0	2.5
Precio rin [$]	0	0	0	0.5	-	0.5	0.5	0.5	0.5	0	2.5
Precio cámara [$]	0	0	0	0.5	0.5	-	0.5	0.5	0.5	0	2.5
Precio neumático [$]	0	0	0	0.5	0.5	0.5	-	0.5	0.5	0	2.5
Precio buje [$]	0	0	0	0.5	0.5	0.5	0.5	-	0.5	0	2.5
Precio del seguro del buje [$]	0	0	0	0.5	0.5	0.5	0.5	0.5	-	0	2.5
Precio total [$]	0	0	0	1	1	1	1	1	1	-	6

Por la calificación total de la tabla anterior se va a ordenar la tabla para realizar el método de objetivos ponderados.

El número más alto corresponderá al 100 y respecto a esto, se irán ponderando las demás posiciones, dándole un valor de 1 al número más bajo.

Ejemplo:

9 – 10
8 – X
$X = \frac{8(10)}{9} = 8.8888 \approx 9$…ecuación 15

Los objetivos se ordenarán como se muestra en la tabla 31 con sus respectivos pesos ponderados:

Tabla 31. Pesos ponderados de la lista de objetivos de las ruedas, rines, cámaras, neumáticos, bujes y seguros del buje.

Objetivos	Meta por alcanzar	Peso ponderado
Tipo del rin	Rin tipo Endrick	8
Tamaño	Máximo diámetro posible de rueda sin descuidar la compatibilidad con el motor.	9
Ancho interior del rin [mm]	Ancho del rin de 37 mm	10
Precio de la rueda [$]	Bajo costo	3
Precio rin [$]	Bajo costo	3
Precio cámara [$]	Bajo costo	3
Precio neumático [$]	Bajo costo	3
Precio buje [$]	Bajo costo	3
Precio del seguro del buje [$]	Bajo costo	3
Precio total [$]	Bajo costo	7

La suma del peso ponderado de todos los objetivos debe dar en total un valor de uno.
En la tabla 32 se presentan las calificaciones según los objetivos de cada uno de los motores siguiendo una escala de once puntos, también se muestra los pesos que se les dieron a los objetivos según su nivel de importancia.

Tabla 32. Calificación de las ruedas, rines, cámaras, neumáticos, bujes y seguros del buje por cada objetivo siguiendo una escala de once puntos.

Objetivo	Peso	A	B	C	D	E
Tipo del rin	8	8	8	8	3	3
Tamaño	9	6	6	6	5	6
Ancho interior del rin [mm]	8	8	8	8	0	8
Precio de la rueda [$]	3	9	9	9	9	9
Precio rin [$]	3	9	9	9	4	6
Precio cámara [$]	3	9	9	9	7	7
Precio neumático [$]	3	9	9	9	5	5
Precio buje [$]	3	9	9	9	4	4
Precio del seguro del buje [$]	3	9	9	9	9	9

Precio total [$]	7	8	10	7	4	5

Cada calificación se multiplicó por el peso que se le dio a cada objetivo y se sumó en su correspondiente columna para obtener el valor general de utilidad, como se muestra en la tabla 33.

Tabla 33. Valor general de utilidad calculado según el de la rueda, rin, cámara, neumático, buje y seguro del buje.

Rueda, rin, cámara, buje y seguro del buje	**A**	**B**	**C**	**D**	**E**
Valor general de utilidad	400	414	393	211	295

Como se muestra en la tabla 33, las dos mejores opciones son la A y la B (figura 22), que son ruedas de bicicleta ya ensambladas; incluyen rin, cámara, neumático, buje y seguro de buje, se decide de entre ellas 2 la mejor tomando en cuenta su precio y aspecto estético, la elección final es la rueda B:

Rueda A:

Color: Negro.
Marca: Benotto.
Precio: $1,049.
Modelo: MTB Aluminio.

Rueda B:

Color: Negro con rojo.
Marca: Kylin.
Precio: $899.
Modelo: MTB Aluminio.

Figura 22. Elección de rueda para el proyecto.

Por la compatibilidad con el motor previamente seleccionado y un mayor puntaje en el diagrama morfológico, se decidió utilizar la rueda **A** para desarrollar el proyecto de la rueda frontal de bicicleta generadora de energía eléctrica.

b.6. Seleccionar la batería portable

Para elegir la batería portable se seleccionará aquella batería que tenga una capacidad de carga de 5000 mAh. También se tomará en cuenta la corriente de entrada, tensión de entrada, precio y peso.

Las baterías con características que son adecuadas para este proyecto son las descritas en la tabla 34:

Tabla 34. Comparación de alternativas de baterías VS objetivos.

	A	B	C	D	E	F	G
Tensión de entrada [V]	5	5	5	5	5	5	5
Corriente de entrada [A]	2.1	2	1	2	2	2	2
Precio [$]	224	299	459	139	349	599	189
Peso (kg)	0.108	0.150	0.136	0.130	0.132	0.150	0.130

Se asignó una calificación a los objetivos más importantes en la tabla 35:

Tabla 35. Ordenar la lista de objetivos de la batería.

Objetivos	Tensión de entrada	Corriente de entrada	Precio	Peso (masa)	Calificación total
Tensión	-	0.5	1	1	2.5
Corriente	0.5	-	1	1	2.5
Precio	0	0	-	0.5	0.5
Peso (masa)	0	0	0.5	-	0.5

Por la calificación total de la tabla anterior se va a ordenar la tabla para realizar el método de objetivos ponderados.

Los objetivos se ordenarán de la manera mostrada en la tabla 36 con sus respectivos pesos ponderados:

Tabla 36. Pesos ponderados de la lista de objetivos del motor.

Objetivos	Meta por alcanzar	Peso ponderado
Tensión de entrada	Tensión de entrada deseada	0.35
Corriente de entrada	Corriente de entrada deseada	0.35
Costo	Peso mínimo	0.15
Peso (kg)	Bajo costo	0.15

La suma del peso ponderado de todos los objetivos debe dar en total un valor de uno.
En la tabla 37 se presentan las calificaciones según los objetivos de cada una de las baterías siguiendo una escala de once puntos, también se muestra los pesos que se les dieron a los objetivos según su nivel de importancia.

Tabla 37. Calificación de los baterías por cada objetivo siguiendo una escala de once puntos.

Objetivo	Peso	A	B	C	D	E	F	G
Tensión	0.35	6	6	6	6	6	6	6
Corriente	0.35	4	6	4	6	6	6	6

Peso [kg]	0.15	6	6	6	6	6	6	6
Costo	0.15	7	7	4	7	5	4	7

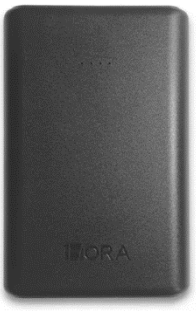
Cada calificación se multiplico por el peso que se le dio a cada objetivo y se sumó en su correspondiente columna para obtener el valor general de utilidad, como se muestra en la tabla 38.

Tabla 38. Valor general de utilidad calculado según la batería.

Batería	**A**	**B**	**C**	**D**	**E**	**F**	**G**
Valor general de utilidad	5.45	6.15	5.00	6.15	5.85	5.70	6.15

Hay tres baterías con mayor valor, las baterías **B, D** y **G**, sus características se muestran en la figura 23:

B
Capacidad de carga: 5000 mAh
Tensión de entrada: 5 V
Corriente de entrada 2 A
Puertos de entrada: 1xMicro USB
Puertos de salida: 2xUSB tipo A
Peso: 0.150 kg
Precio: $299
Color: Blanco/Negro
Indicador de carga: Si



D
Capacidad de carga: 5000 mAh
Tensión de entrada:5 V
Corriente de entrada 2 A
Puertos de entrada: 1xMicro USB y 1x USB tipo C
Puertos de salida: 1xUSB tipo A
Peso: 0.130 kg
Precio: $139
Color: Blanco/Negro
Indicador de carga: Si

G
Capacidad de carga: 5000 mAh
Tensión de entrada: 5 V
Corriente de entrada: 2 A
Puertos de entrada: 1xMicro USB
Puertos de salida: 2xUSB tipo A
Peso: 0.130 kg
Precio: $189
Color: Blanco/Negro
Indicador de carga: Si

Figura 23. Elección de batería portable para el proyecto.

Comparando ambas baterías podemos observar que las tres cumplen con los objetivos principales que es la capacidad de carga, tensión de entrada y corriente de entrada. Para seleccionar una, dependerá del precio, por lo que se escogerá la batería **D**, por ser la de menor precio.

b.7. Seleccionar los dispositivos electrónicos programables que lleven a cabo la comunicación inalámbrica entre la rueda y el dispositivo móvil

Para realizar la selección de un dispositivo de control se tomó en cuenta que el dispositivo de control ya tenga integrado un módulo de comunicación wifi y bluetooth. Además de eso se tomaron en cuenta los siguientes objetivos importantes para seleccionar el dispositivo de control (tabla 39).

Tabla 39. Calificación de objetivos.

Objetivos	Consumo Energético	Wifi	Bluetooth	Puertos Analógicos Lectura	Peso	Precio	Suma
Consumo Energético		0	0	1	1	1	3
Wifi	0.5		0.5	1	1	0.5	3.5
Bluetooth	0.5	0.5		1	1	0.5	3.5
Puerto Analógico	0	0	0		0.5	0	0.5
Peso	0	0	0	0.5		0.5	1
Precio	0	0	0	0.5	0.5		1

Se calificaron los objetivos para ordenarlos del más importante a menos importante.

En la tabla 40 se muestran los datos de los dispositivos controladores adecuados para este proyecto, y sus características según los objetivos.

Tabla 40. Datos de dispositivos de control según los objetivos.

Objetivos	RF Nano V3.0 ATmega328P	Raspberry Pi 3	SparkFun ESP32	ESP32
Wifi	0	1	1	1
Bluetooth	0	1	0	1
Consumo Energético	0.095	2.75	0.05	0.25
Peso	9	50	22.7	6.8
Precio	356.95	1389	508.18	270
Puerto Analógicos	1	0	1	1

En la tabla 41, se muestra en a la derecha su valor sumado, en este caso se le asignaron los siguientes pesos de calificación a cada objetivo, en total suman uno.

Tabla 41. Pesos ponderados de la lista del dispositivo de control.

Objetivos	Meta por alcanzar	Peso ponderado
Wifi	Incluir comunicación Wifi	0.25
Bluetooth	Incluir comunicación Bluetooth	0.25
Consumo energético	Bajo consumo energético	0.15
Peso	Bajo peso	0.125
Precio	Bajo costo	0.125
Lectura analógica	Incluir puerto de lectura analógica	0.1

En la tabla 42 se presentan las calificaciones según una escala del 0 al 10, después de esa calificación de cada objetivo del microcontrolador se procedió a multiplicar con el peso. Por ultimo los valores de cada dispositivo de control se sumaron.

Tabla 42. Calificación y valor para seleccionar dispositivo de control.

Objetivo	Peso	RF Nano V3.0 ATmega328P Calificación:	Valor	Raspberry Pi 3 Calificación:	Valor	SparkFun ESP32 Calificación:	Valor	ESP32 Calificación:	Valor
Wifi	0.25	3	0.75	8	2	8	2	8	2
Bluetooth	0.25	3	0.75	8	2	3	0.75	8	2
Consumo Energético	0.15	5	0.75	3	0.45	8	1.2	5	0.75
Peso	0.125	8	1	3	0.375	4	0.5	8	1
Precio	0.125	5	0.625	3	0.375	4	0.5	7	0.875
Puerto Analógicos	0.1	8	0.8	3	0.3	8	0.8	8	0.8
			4.68		5.5		5.75		7.42

Como se puede observar el dispositivo de control ganador es el ESP32 (figura 24) por el cual cuenta con las dos tecnologías de comunicación que se necesitan, además de que su consumo energético es aceptable en comparación con otros dispositivos, tomando en cuenta también que el precio es menor.

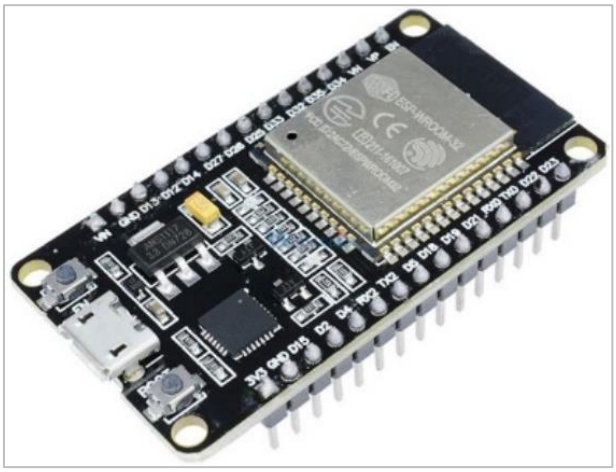


Figura 24. ESP32, dispositivo de control seleccionado.

c. Diseño de materialización

En el diseño de materialización se desarrollan los aspectos iniciales a tomarse en cuenta para diseñar el proyecto, tomando en cuenta los posibles métodos, elementos o materiales a utilizar.

c.1. Estructura mecánica

Para el diseño de materialización de la estructura mecánica se tomarán en cuenta las dimensiones importantes de las distintas piezas del proyecto, para que coincidan con la bicicleta de pruebas y que así se pueda determinar el material a utilizar para manufacturar cada pieza que compone en el ensamble de la estructura.

c.1.1. Material base

La bicicleta que se usó para diseñar la estructura y posteriormente se utilizará para pruebas, es de tipo MTB, marca Monk Kron rodada 26 de 18 velocidades (figuras 25 y 26), con marco es de acero, rines de aluminio tipo Sprint y de 36 radios [34].

Figura 25. Bicicleta de pruebas Monk Kron R26 18 V.

Figura 26. Bicicleta Monk Kron con marco de acero y 18 velocidades.

Los datos importantes para tomar en cuenta en el diseño de la estructura mecánica son:

La distancia de separación entre los eslabones de la horquilla es de 108 mm, además el mismo motor MXUS-XF07 está hecho para colocarse en esa parte, por lo que su ancho también es de 108 mm.

El diámetro interior del rin MTB de Aluminio marca Kylin es de 545 mm.

Se pretende que el usuario de la rueda frontal de bicicleta generadora de energía pueda introducir y retirar la batería portable de la estructura, por lo que hay que tomar en cuenta las dimensiones máximas de una batería portable, el mayor tamaño encontrado por los autores del presente trabajo es el del cargador portátil Dell Power Bank Pw7015l de 18,000mAh con dimensiones de 21.5 X 162 X 78 mm [35], por lo que estas dimensiones tendrán que ser tomadas en cuenta para crear el compartimento de la batería portable y definir la separación entre las placas laterales de la estructura mecánica dada por el buje de la estructura.

c.1.2. Acoplamiento mecánico entre los generadores y el rin de la rueda

Han sido adquiridos y desmontados el rin, buje, neumático y cámara de la rueda, la rueda adquirida es de tipo Endrick, también se ha obtenido el motor MXUS-XF07 que funcionará como generador eléctrico (figura 27) y ambos han sido modelados en CAD (figuras 28, 29 y 30) para poder diseñar el riel que mantendrá alineado el rin con el motor, asegurando así que exista fricción y estabilidad entre ellos mientras la estructura se mantiene estática durante la rotación de la rueda, ya que estará sujetada a la horquilla de la bicicleta. Los modelos se mostraron usando la herramienta de visualización Photoview 3D de SolidWorks, que permite ver los modelos de una forma más estética.

Figura 27. Fotografías del motor MXUS – XF07 adquirido.

Figura 28. Modelado 3D en SolidWorks del motor MXUS – XF07 utilizando la herramienta de visualización Photoview 3D.

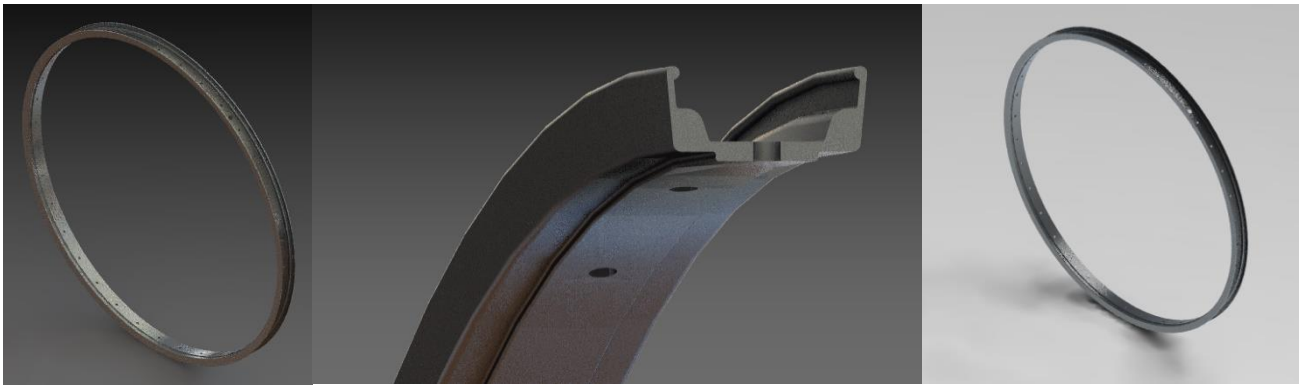


Figura 29. Modelado 3D en SolidWorks del rin con 36 radios utilizando la herramienta de visualización Photoview 3D.

Figura 30. Modelado 3D en SolidWorks del rin y llanta utilizando la herramienta de visualización Photoview 3D.

Para crear el riel de la rueda sobre el rin se usará una correa de transmisión mecánica con perfil trapezoidal tipo C, B y Z dentada (figura 31) porque el ancho del rin es de 29.5 mm y esos tres perfiles son los que más se acercan a tener el ancho exterior necesario sin pasarse, permitiendo por su altura también que pase la boquilla de la cámara en la rueda frontal. En el **Apéndice A**, se muestra el catálogo de la empresa CORREAS BELSA donde se encuentra la longitud externa exacta que se busca,

considerando que el diámetro interno del rin es de 545 mm y su perímetro es de 1712.1679, el modelo de correa elegida es el C64 con longitud externa de 1714 mm [36]:

Figura 31. Riel usando una banda trapezoidal perfil C64 para la rueda frontal generadora de energía.

c.1.3. Análisis cinemático

De acuerdo con el ranking de ciclociudades edición 2018, la velocidad promedio para transitar en ciudades mexicanas es de 20 km/h [44].

Se utiliza la ecuación de velocidad para el análisis de movimiento relativo obtenida de [40] para obtener los vectores que describen las velocidades en la rueda obteniendo además su dirección, considerando que en la rueda se tienen dos tipos de movimientos distintos: El movimiento rotacional de la rueda, cuyo centro instantáneo de velocidad cero se encuentra en el punto donde la llanta tiene contacto con el suelo, debido a que el suelo no está en movimiento y el movimiento de traslación de las placas laterales y los elementos que estén en contacto con ella, ya que esta parte de la rueda no girará durante el uso de la rueda frontal de bicicleta generadora de energía eléctrica.

Datos:

$D_{R1} = Diámetro\ de\ la\ polea = 32\ mm$ …ecuación 16

$D_{int} = 517\ mm$…ecuación 17

$D_{ext} = 26" = 660.4\ mm$…ecuación 18

$A_{ll} = Ancho\ llanta = 71.7\ mm$ …ecuación 19

$D_{motor} = 121\ mm$…ecuación 20

$V_{bici} = 20\frac{km}{hr}$…ecuación 21

$CI = Centro\ instantáneo\ de\ velocidad\ cero$ …ecuación 22

$W_m = Velocidad\ angular\ del\ motor$ …ecuación 23

$W_R = Velocidad\ angular\ rueda$ …ecuación 24

Producto cruz de los vectores unitarios de dirección $\hat{\imath}$, $\hat{\jmath}$ y $\hat{k}$:

$\hat{\imath}\ x\ \hat{\imath} = 0$…ecuación 25

$\hat{\jmath}\ x\ \hat{\jmath} = 0$…ecuación 26

$\hat{k}\ x\ \hat{k} = 0$…ecuación 27

$\hat{\imath}\ x\ \hat{\jmath} = \hat{k}$…ecuación 28

$\hat{\jmath}\ x\ \hat{k} = \hat{\imath}$…ecuación 29

$\hat{k}\ x\ \hat{\imath} = \hat{\jmath}$…ecuación 30

$\hat{\jmath}\ x\ \hat{\imath} = -\hat{k}$…ecuación 31

$\hat{k}\ x\ \hat{\jmath} = -\hat{\imath}$…ecuación 32

$\hat{\imath}\ x\ \hat{k} = -\hat{\jmath}$…ecuación 33

Velocidades de los generadores fuera del eje de rotación:

$V_{Bici} = 20\left[\frac{km}{hr}\right]\left[\frac{1000m}{1\ km}\right]\left[\frac{1\ hr}{3600\ seg}\right] = 5.5555\left[\frac{m}{s}\right]$…ecuación 34

$V_{Bici} = V_{CI} + (w_{rueda}\ X\ r_{CI}^{r}) = 0 + \left(-w_{rueda}[\hat{k}]\ X\ \frac{0.6604}{2}[\hat{\jmath}]\right) = -0.302(-\hat{\imath}) = 5.5555\left[\frac{m}{s}\right]$…ecuación 35

$w_{rueda} = 16.8246\left[\frac{rad}{s}\right] = 16.8246\left[\frac{rad}{s}\right]\left[\frac{1\ rev}{2\pi\ rad}\right]\left[\frac{60\ seg}{1\ min}\right] = 160.663\ [rpm]$…ecuación 36

$V_A = V_{CI} + (w_{rueda}\ X\ r_{CI}^{A}) = 0 + \left(-16.8246[\hat{k}]\ X\ (0.0717 + 0.517)[\hat{\jmath}]\right) = -9.9046[-\hat{\imath}] = 9.9046\left[\frac{m}{s}\right]$…ecuación 37

$V_B = V_{CI} + (w_{rueda}\ X\ r_{CI}^{B}) = 0 + \left(-16.8246[\hat{k}]\ X\ \left(\left(0.242 + \frac{0.032}{2}\right)[\hat{\imath}] + \left(\frac{0.6604}{2}\right)[\hat{\jmath}]\right)\right) = 5.5555[\hat{\imath}] - 4.3407[\hat{\jmath}]\left[\frac{m}{s}\right]$…ecuación 38

$V_C = V_{CI} + \left(w_{rueda}\ X\ r_{CI}^{C}\right) = 0 + \left(-16.8246[\hat{k}]\ X\ 0.0717[\hat{\jmath}]\ \right) = 1.2063[\hat{\imath}]\left[\frac{m}{s}\right]$…ecuación 39

$V_D = V_{CI} + (w_{rueda}\ X\ r_{CI}^{D}) = 0 + \left(-16.8246[\hat{k}]\ X\ \left(-\left(0.242 + \frac{0.032}{2}\right)[\hat{\imath}] + \left(\frac{0.6604}{2}\right)[\hat{\jmath}]\right)\right) = 5.5555[\hat{\imath}] + 4.3407[\hat{\jmath}]\left[\frac{m}{s}\right]$…ecuación 40

Velocidades angulares de los generadores y ruedas laterales:

$V_A = V_{Bici} + (w_{m1}\ X\ r_{m1}^{A}) = 5.5555[\hat{\imath}] + \left(w_{m1}[-\hat{k}]\ X\ \frac{0.121}{2}[\hat{\jmath}]\right) = 9.9046[\hat{\imath}]$…ecuación 41

$w_{m1} = 71.8859\left[\frac{rad}{s}\right] = 71.8859\left[\frac{rad}{s}\right]\left[\frac{1\ rev}{2\pi\ rad}\right]\left[\frac{60\ seg}{1\ min}\right] = 686.4597\ [rpm] = 71.8859\left[\frac{rad}{s}\right]\left[\frac{180\ deg}{\pi\ rad}\right] = 4118.7586\left[\frac{deg}{s}\right]$…ecuación 42

$V_C = V_{Bici} + (w_{m2}\ X\ r_{m2}^{C}) = 5.5555[\hat{\imath}] + \left(w_{m2}[-\hat{k}]\ X\ \frac{-0.121}{2}[\hat{\jmath}]\right) = 1.2063[\hat{\imath}]\left[\frac{m}{s}\right]$…ecuación 43

$w_{m2} = 71.8876\left[\frac{rad}{s}\right] = 71.8876\left[\frac{rad}{s}\right]\left[\frac{1\ rev}{2\pi\ rad}\right]\left[\frac{60\ seg}{1\ min}\right] = 686.4760\ [rpm] = 71.8876\left[\frac{rad}{s}\right]\left[\frac{180\ deg}{\pi\ rad}\right] = 4118.856\left[\frac{deg}{s}\right]$…ecuación 44

$V_D = V_{Bici} + (w_R\ X\ r_R^{D}) = 5.5555[\hat{\imath}] + \left(w_R[-\hat{k}]\ X\ \frac{-0.032}{2}[\hat{\jmath}]\right) = 5.5555[\hat{\imath}] + 4.3407[\hat{\jmath}]\left[\frac{m}{s}\right]$…ecuación 45

$\hat{\imath}$: $5.5555 = 5.5555$…ecuación 46

$\hat{\jmath}$: $w_R = 271.2937\left[\frac{rad}{s}\right] = 271.2937\left[\frac{rad}{s}\right]\left[\frac{1\ rev}{2\pi\ rad}\right]\left[\frac{60\ seg}{1\ min}\right] = 2590.664\ [rpm] = 271.2937\left[\frac{rad}{s}\right]\left[\frac{180\ deg}{\pi\ rad}\right] = 15{,}543.984\left[\frac{deg}{s}\right]$…ecuación 47

Si los generadores se colocan alrededor del rin a distintos ángulos de inclinación respecto al eje vertical se mantiene una rotación de 72 $\left[\frac{rad}{s}\right]$ aproximadamente, no importando donde se encuentren, pero si se compara esta rotación con la que tendría el generador si se colocara justo en el eje de rotación de la rueda que transita a 20 km/h se obtiene lo siguiente al hacer el análisis cinemático.

Velocidades generador situado en el eje de rotación:

$w_{rueda} = 16.8246\left[\frac{rad}{s}\right] = 16.8246\left[\frac{rad}{s}\right]\left[\frac{1\,rev}{2\pi\,rad}\right]\left[\frac{60\,seg}{1\,min}\right] = 160.663\,[rpm]$…ecuación 48

$V_A = V_{CI} + (w_{rueda}\,X\,r_{CI}^{A}) = 0 + \left(-16.8246[\hat{k}]\,X\,\left(0.0717 + \frac{0.517}{2} + \frac{0.121}{2}\right)[\hat{\jmath}]\right) = -6.5733[-\hat{\imath}] = 6.5733\left[\frac{m}{s}\right]$…ecuación 49

Velocidad angular del generador:

$V_A = V_{Bici} + (w_{m1}\,X\,r_{m1}^{A}) = 5.5555[\hat{\imath}] + \left(w_{m1}[-\hat{k}]\,X\,\frac{0.121}{2}[\hat{\jmath}]\right) = 6.5733[\hat{\imath}]$…ecuación 50

$w_{m1} = 16.8231\left[\frac{rad}{s}\right] = 16.8231\left[\frac{rad}{s}\right]\left[\frac{1\,rev}{2\pi\,rad}\right]\left[\frac{60\,seg}{1\,min}\right] = 160.6487\,[rpm]$…ecuación 51

Haciendo la comparación entre las velocidades angulares obtenidas se puede concluir que el colocar el generador fuera del eje de rotación de la rueda se obtiene la siguiente ganancia A_w por cada generador, que equivale a ser cuatro veces mayor al giro que tendría el generador si se colocara en el eje de rotación de la rueda, por lo cual si se están utilizando 2 generadores dentro de la rueda frontal, se estaría generando ocho veces más energía que si se colocara el generador en el eje de rotación de la rueda frontal de bicicleta:

$A_w = \frac{686.4760\,[rpm]}{160.6487\,[rpm]} = 4.2731 \approx 4$…ecuación 52

$2A_w = 2(4.2731) \approx 8$…ecuación 53

c.1.5. Lámina lateral de la estructura mecánica

Para diseñar la estructura mecánica en el interior de la rueda frontal se deben tener las siguientes consideraciones al realizar el análisis dinámico usando la segunda ley de Newton:

- Al iniciar un recorrido en bicicleta, lo más probable es que el usuario incline su cuerpo hacia adelante para avanzar y si su peso máximo es de 100 kg, todo ese peso caerá sobre el eje (o buje) de la rueda frontal a través de la horquilla, que se encuentra en contacto con él por medio de sus dos brazos a un ángulo de inclinación de 25°. El ángulo fue medido con un compás por los miembros del equipo ya que se cuenta en este momento con la bicicleta de pruebas y con ella se realizó su modelo CAD con medidas reales para realizar el diseño de la estructura mecánica.

- Cada motor pesa aproximadamente 3 kg (figura 32), este peso también será soportado por la estructura mecánica de la rueda frontal, por lo que deberá ser tomado en cuenta como fuerza externa.

- Los motores deberán estar alineados verticalmente para que toda la fuerza aplicada sea contrarrestada por la estructura, ya que, si no, se corre el riesgo de que en el trayecto de la bicicleta el rin frontal se deforme.

- De forma perpendicular a los motores existirán dos ruedas laterales en forma de polea que funcionan como soporte y estarán en contacto con la banda del rin, equilibrando además la componente horizontal de la fuerza aplicada por el peso del usuario a través de la horquilla sobre

el buje de la estructura, estos elementos a su vez deben rotar para permitir el giro de los motores durante el trayecto de la bicicleta mientras la estructura se mantiene estática. Para crear el eje de rotación de estas ruedas se utilizarán remaches.

- Como el análisis es dinámico, se debe contar con la aceleración del recorrido, para ello se considerará que se requiere alcanzar las revoluciones indicadas en la figura 13 para alcanzar una tensión de 24.8 V y corriente de 3.7A en cada generador. Para el cálculo se propone que el usuario llegue de 0 a 140 rpm en 15 segundos y que la rueda gira sin deslizamiento.

- La estructura mecánica que soporta todo el sistema se conforma de dos placas paralelas unidas por remaches, esto se realizó así para poder manufacturar con aluminio y reducir el peso de la estructura, además de proporcionar una manufactura fácil y sencilla, ya que el presente proyecto es patrocinado y se busca llevarlo al mercado, por lo que una manufactura sencilla y con poco peso es la solución ideal.

- El peso total del sistema se estima que sea de 120 kg tomando en cuenta el peso de la bicicleta completa, incluyendo la rueda frontal generadora de energía y el peso máximo del usuario de 100 kg.

Figura 32. Peso del motor MXUS-XF07.

De acuerdo con el ranking de ciclociudades edición 2018, la velocidad promedio para transitar en ciudades mexicanas es de 20 km/h [44].

Datos:

$W_{motor} = \frac{3}{2}(9.81) = 14.715\ N$
…ecuación 54

$F_{max} = \frac{100}{2}(9.81) = 490.5\ N$
…ecuación 55

$tiempo\ aceleración = 15\ s$
…ecuación 56

$m_t = 120\ kg = masa\ total$…ecuación 57

$D_{ext} = 26" = 660.4\ mm$…ecuación 58

$V_{bici} = 20\frac{km}{hr}$…ecuación 59

$F_f = Fuerza\ de\ fricción$ …ecuación 60

$F_{R1} = Reacción\ en\ la\ polea$ …ecuación 61

$N = Normal$…ecuación 62

Cálculos dinámicos:

$\sum F_x = m_t(a_x)$ …ecuación 63

$\sum F_y = 0$…ecuación 64

$\sum T = 0$…ecuación 65

$V_{Bici} = 20\left[\frac{km}{hr}\right]\left[\frac{1000m}{1\ km}\right]\left[\frac{1\ hr}{3600\ seg}\right] = 5.5555\left[\frac{m}{s}\right]$…ecuación 66

$V_{Bici} = V_{CI} + (w_{rueda}\ X\ r_{CI}^{r}) = 0 + \left(-w_{rueda}[\hat{k}]\ X\ \frac{0.6604}{2}[\hat{\jmath}]\right) = -0.3302(-i) = 5.5555\left[\frac{m}{s}\right]$…ecuación 67

$w_{rueda} = 16.8246\left[\frac{rad}{s}\right] = 16.8246\left[\frac{rad}{s}\right]\left[\frac{1\ rev}{2\pi\ rad}\right]\left[\frac{60\ seg}{1\ min}\right] = 160.663\ [rpm]$…ecuación 68

$a_x = \frac{Vf-V0}{t} = \frac{5.5555-0}{15} = 0.3703\left[\frac{m}{s^2}\right]$…ecuación 69

$\sum F_x = 490.5\sin 25 + F_f - F_{R1} = 120(0.3703)$ …ecuación 70

$F_{R1} - F_f = 162.8582$…ecuación 71

$\sum F_y = -14.715 * 2 - 490.5\cos 25 + \frac{N}{2} = 0$…ecuación 72

$\frac{N}{2} = 473.9739\ N$…ecuación 73

$N = 947.9479\ N$…ecuación 74

$\sum T = xF_y - yF_x = 0$…ecuación 75

$0 - \left(\left(\frac{0.545}{2}\right)(-F_{R1} + 490.5\sin 25)\right) = 0$…ecuación 76

$F_{R1} = 207.2942\ N$…ecuación 77

$F_f = F_{R1} - 162.8582 = 44.436\ N$…ecuación 78

Cálculos estáticos:

$\sum F_x = 0$…ecuación 79

$\sum F_y = 0$…ecuación 80

$\sum T = 0$…ecuación 81

$\sum F_x = 490.5\sin 25 - F_{R1} = 0$…ecuación 82

$F_{R1} = 207.2942\ N$…ecuación 83

$\sum F_y = -14.715 * 2 - 490.5\cos 25 + \frac{N}{2} = 0$…ecuación 84

$\frac{N}{2} = 473.9739\ N$…ecuación 85

$N = 947.9479\ N$…ecuación 86

Al comparar los cálculos dinámicos y los estáticos se llega a la conclusión de que la única fuerza que varía su valor es la fuerza de fricción F_f que siempre tiene el valor de la masa total del sistema por la aceleración.

Debido a que la estructura aguantará verticalmente una fuerza máxima de 473.9739 N aplicada por el peso límite del usuario y distribuida equitativamente entre los 2 brazos de la horquilla sobre el eje de la

estructura de la rueda frontal, se considera para elegir el material, tomando en cuenta solo el criterio de resistencia, ya que no hay restricciones respecto a las deformaciones de la estructura.

Para encontrar el esfuerzo máximo de la estructura creada con lámina de aluminio, que es donde estará el mayor esfuerzo que la puede llevar a la falla, se tomará en cuenta espesor de la placa y el ancho mínimo lateral de la estructura que es de 53.5 mm para elegir un material en función del factor de seguridad resultante:

$$\sigma_t = \frac{Fmax}{Amin} = \frac{473.9739}{0.0535 * e} \ldots \text{ecuación 87}$$

Debido a que se busca que el peso sea el menor posible, se explorarán las distintas aleaciones y temples de Aluminio con el proveedor Metales Díaz, en función del calibre comercial de la placa descritas en la tabla 1 del **Anexo 5**, creada por los autores del presente trabajo. Observando su catálogo incluido en el **Apéndice B** del presente documento, se nota la gran gama de aleaciones de aluminio que poseen y en función de las siguientes propiedades mecánicas, se debe elegir un material en específico para la construcción de la estructura, siempre y cuando se encuentre disponible como lámina lisa [37]:

- Resistencia a la tensión - Punto de cedencia.
- Corte - Esfuerzo al corte inicial.
- Módulo – Módulo de elasticidad.

Además de mostrarse en forma de tabla, se graficó el calibre de cada material contra el factor de seguridad que implica al hacer el cálculo del criterio de resistencia para da uno (figuras 33, 34, 35, 36, 37, 38 y 39).

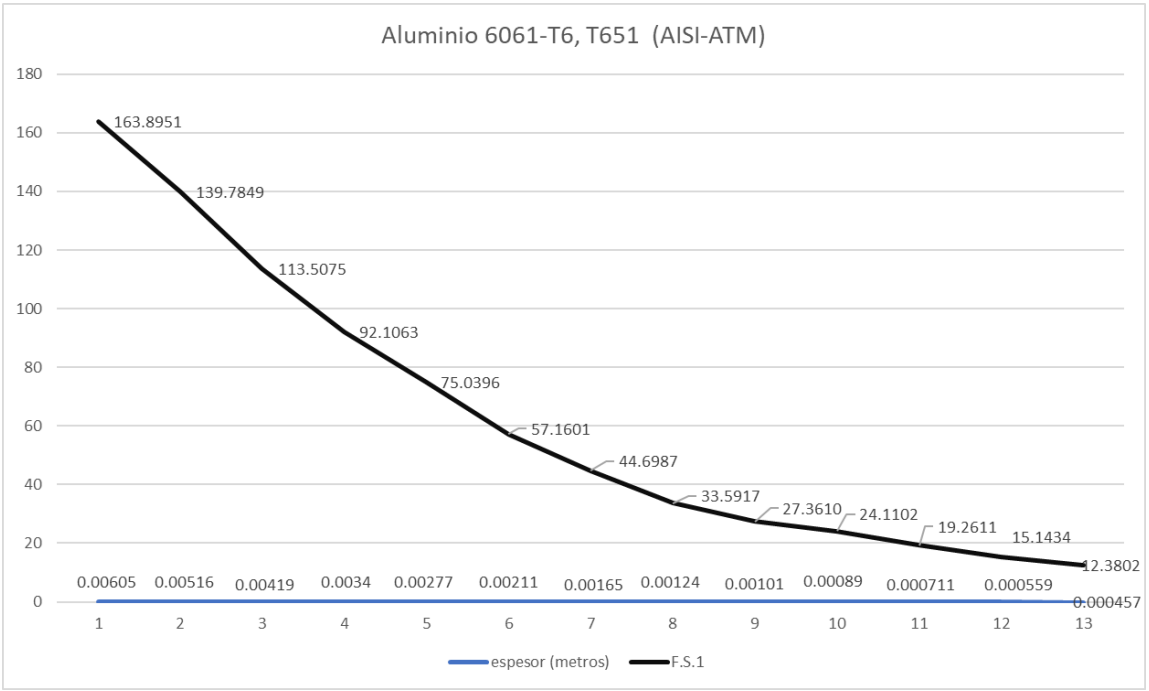


Figura 33. Gráfica de espesor contra factor de seguridad con el Aluminio 6061-T6, T651 (AISI-ASTM).

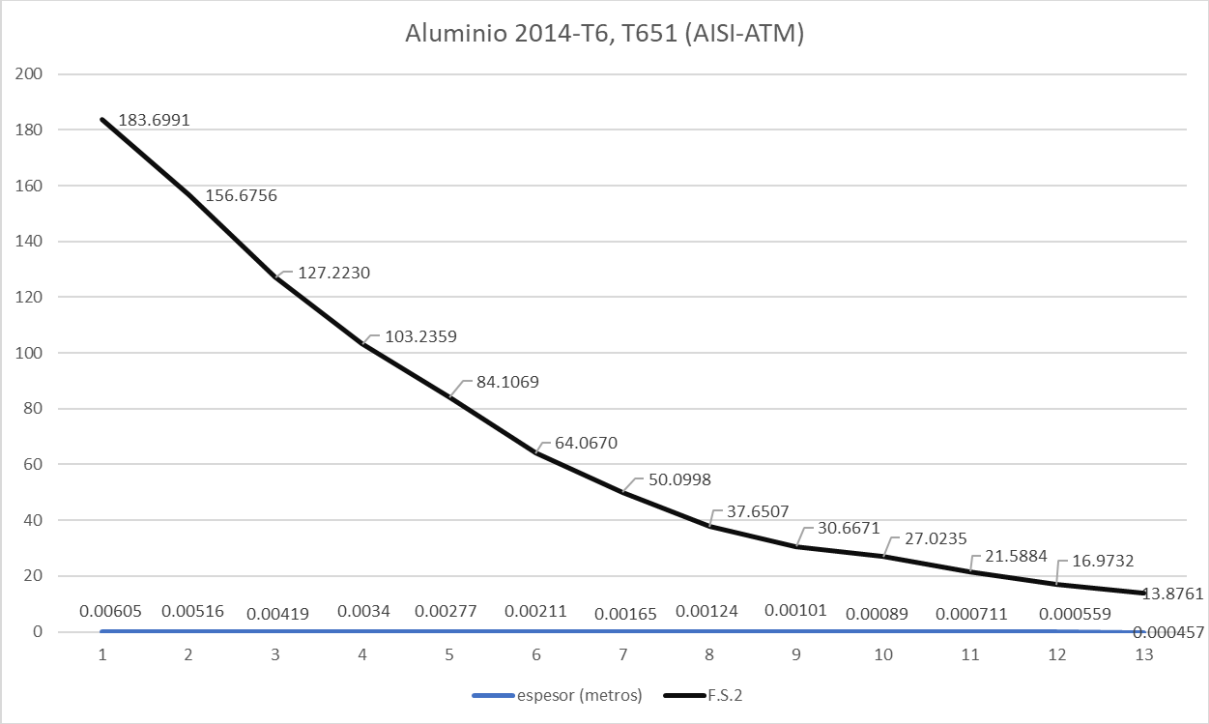


Figura 34. Gráfica de espesor contra factor de seguridad con el Aluminio 2014-T6, T651 (AISI-ASTM).

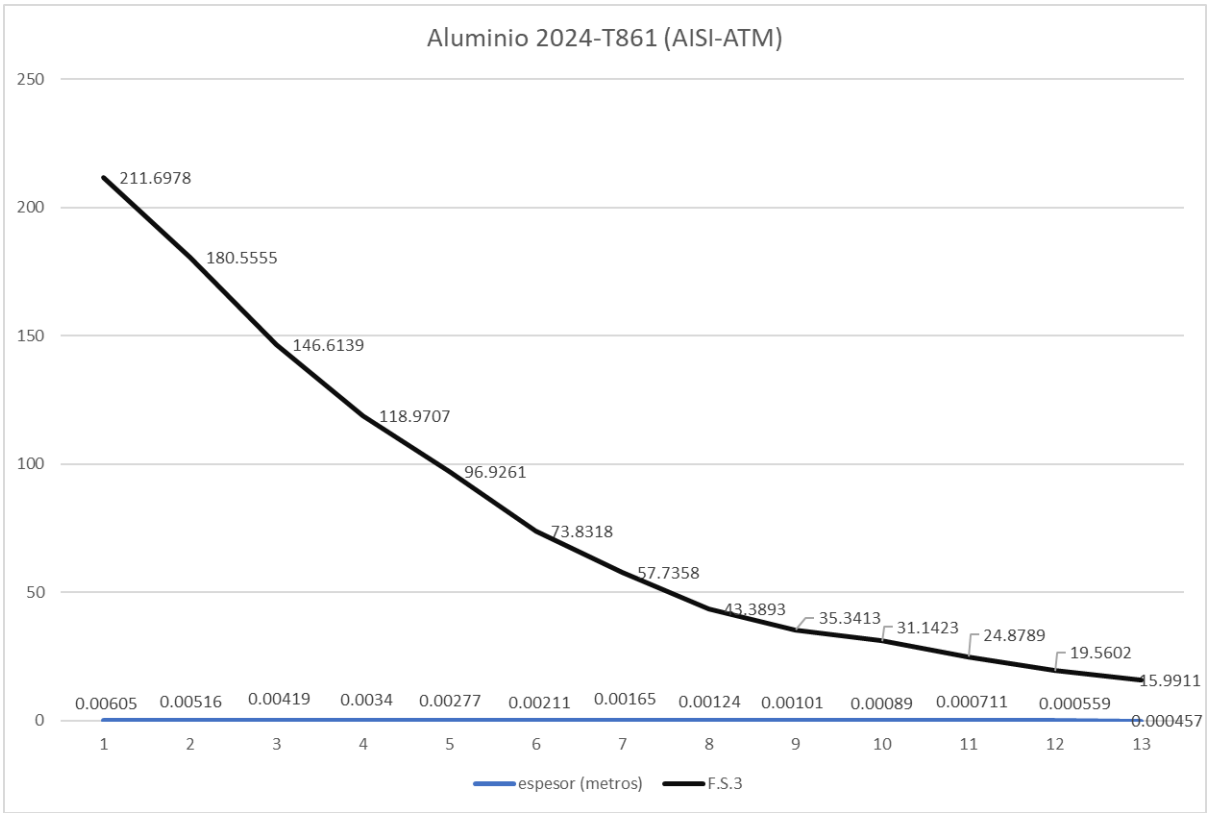


Figura 35. Gráfica de espesor contra factor de seguridad con el Aluminio 2024-T861 (AISI-ASTM).

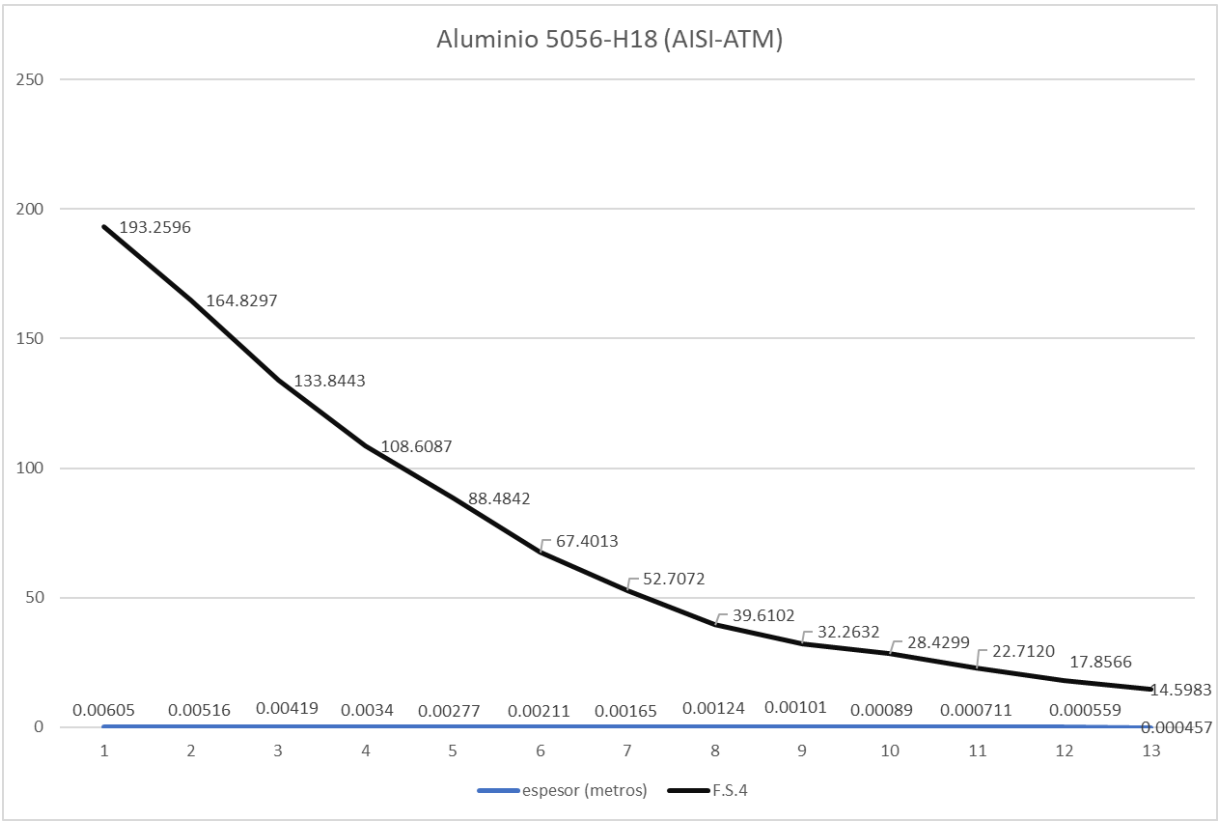


Figura 36. Gráfica de espesor contra factor de seguridad con el Aluminio 5056-H18 (AISI-ASTM).

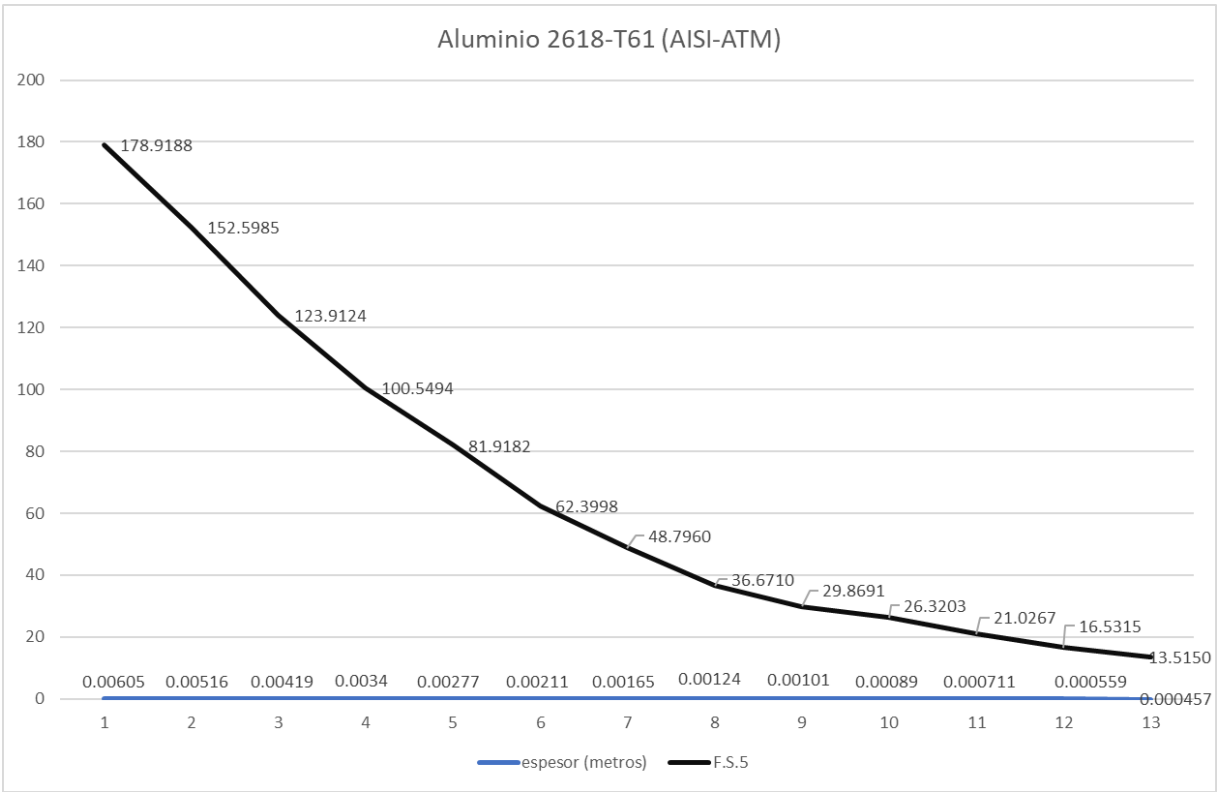


Figura 37. Gráfica de espesor contra factor de seguridad con el Aluminio 2618-T61 (AISI-ASTM).

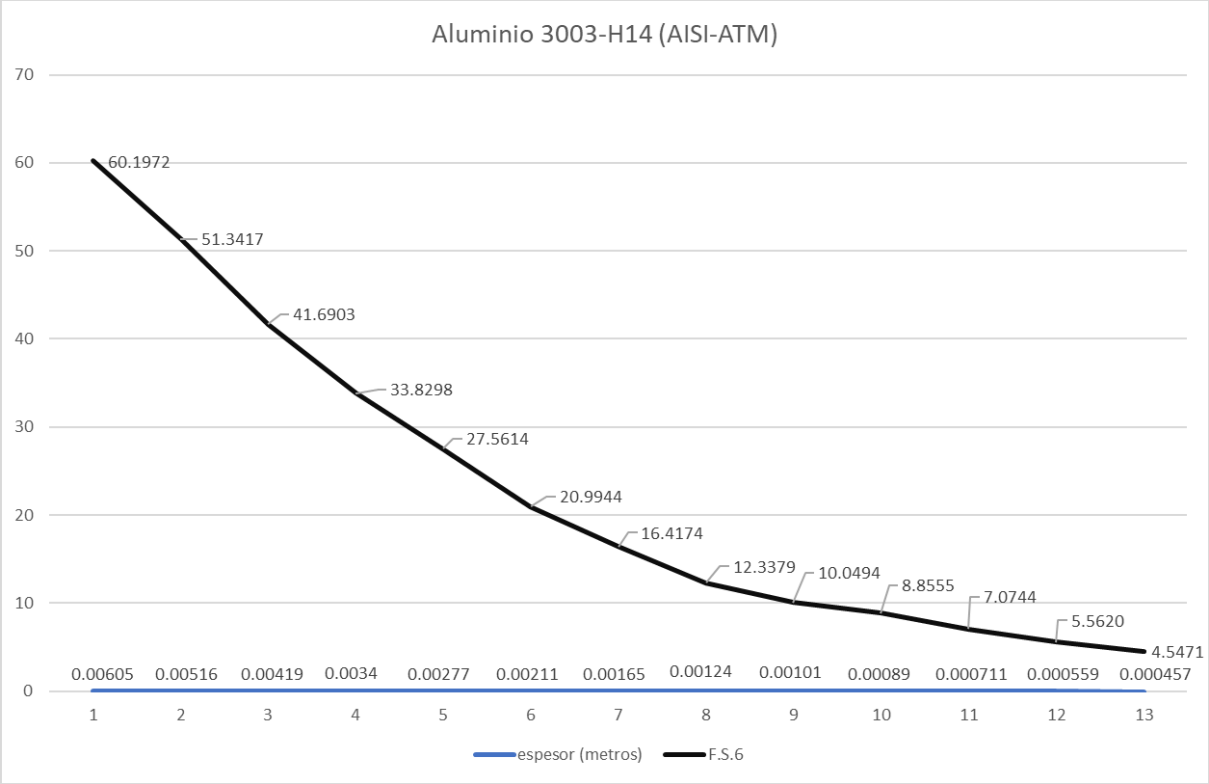


Figura 38. Gráfica de espesor contra factor de seguridad con el Aluminio 3003-H14 (AISI-ASTM).

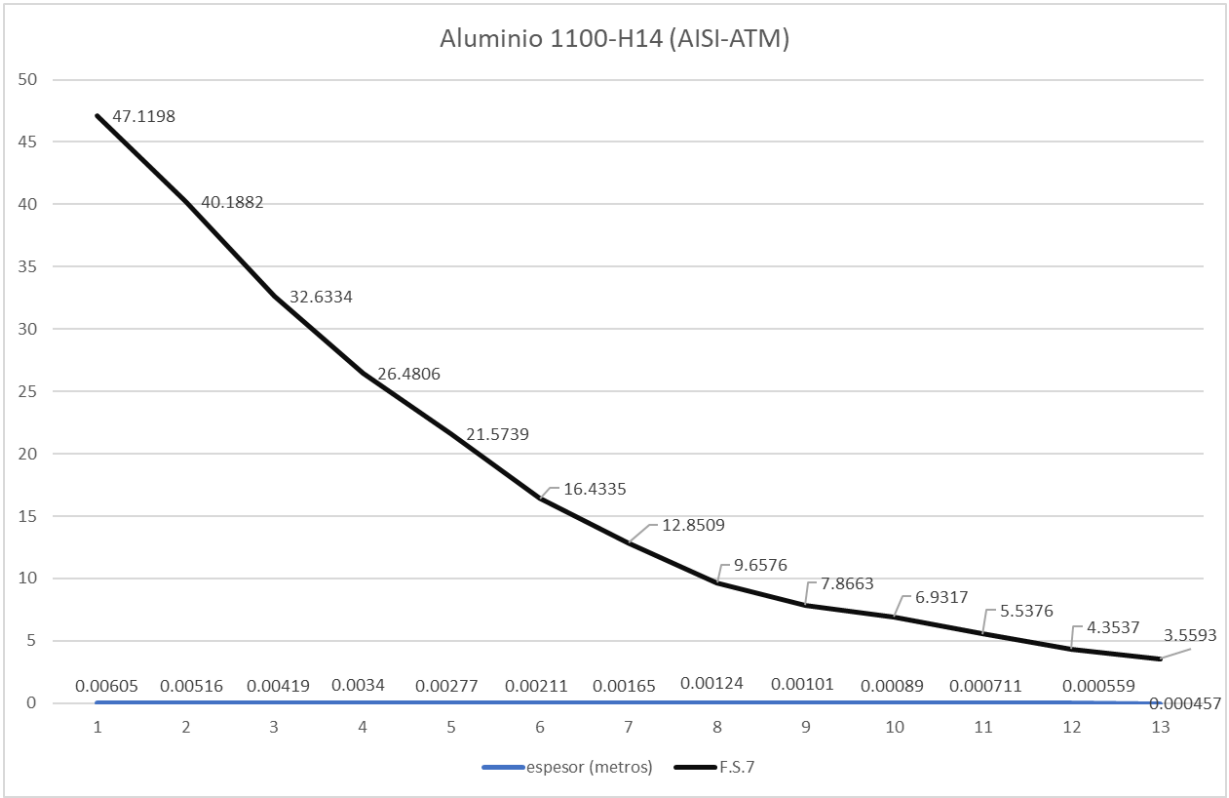


Figura 39. Gráfica de espesor contra factor de seguridad con el Aluminio 1100-H14 (AISI-ASTM).

Ninguno de los calibres de aluminio dio como resultado un factor de seguridad menor a 1, por lo que en teoría cualquiera puede ser elegido para el proyecto. Los dos materiales que están disponibles como lámina de aluminio son los siguientes, como le fue indicado a los autores del presente trabajo por el personal de servicio a clientes perteneciente a la empresa Metales Díaz, mostrado en el **Apéndice C**:

- **Aleación de aluminio 1100-H14:** Cuya composición química consiste en 0.95 Silicio y Fierro, 0.5-0.2 Cobre, 0.05 Manganeso, 0.1 Zinc, 0.05 de Otros y 99.0 Aluminio en su composición química [37].

- **Aleación de aluminio 3003-H14:** Cuya composición química consiste en 0.6 Silicio, 0.7 Fierro, 0.5-0.2 Cobre, 1.0-1.5 Manganeso, 0.1 Zinc, 0.05 de Otros y 85.0 Aluminio en su composición química [37].

Se busca tener factores de seguridad cercanos a 10 para que el diseño sea robusto desde una primera instancia, ya que se pretende agregar aberturas estéticas en la estructura mecánica, creando concentradores de esfuerzo y reduciendo así el factor de seguridad encontrado en este momento, por lo que se elegirán los calibres 10, 12 y 14 con espesores de 3.4, 2.77 y 2.11 mm considerando como opción lámina lisa de aluminio de los materiales 1100-H14 y 3003-H14.

c.1.6. Buje de la estructura mecánica

De igual manera se realiza un cálculo analítico para determinar el material de la barra redonda con el cual se va a diseñar el buje de la estructura, al buje se ensamblarán las placas laterales por medio de remaches, para ello se considera que el eje debe tener un diámetro exterior de 12 mm y un recorte de 10 mm de ancho a través del mismo diámetro para que se pueda introducir correctamente en la horquilla, además debe contar con un diámetro interior de 5 mm para que dentro de él se pueda introducir el eje de cierre rápido que tiene un diámetro exterior de 4.5 mm, el eje de cierre rápido se integra al buje para que el usuario pueda colocar y remover la rueda de forma sencilla de cualquier bicicleta sin necesidad de herramientas, ya que este se ajusta por medio de una perilla y resortes internos.

Al realizar el cálculo analítico se toma en cuenta que la longitud total del buje debe ser de 108 mm para que coincida con la horquilla de la bicicleta de pruebas Monk Kron, además se analizará la opción de que tenga una separación entre las placas de 35 o 50 mm, ya que es suficiente que el espacio entre las placas con tal que sea mayor a 21.5 mm porque ese fue el espesor máximo de una batería removible encontrado.

La fuerza aplicada sobre el buje es del peso máximo del usuario que es 100 kg entre 2 ya que la horquilla cuenta con dos brazos a través de los cuales se distribuye la carga equitativamente debido a que son simétricas, la fuerza está inclinada 25° y se descompone en su componente vertical y horizontal, dando como resultado dos esfuerzos que someten a flexión al buje en distintas direcciones, para la comprobación de los diagramas de fuerza cortante y momento flexionante se utilizó una herramienta web llamada VIGA Online (de la figura 40 hasta la 51) [39].

Datos:

$F_{max} = \frac{100}{2}(9.81) = 490.5\ N$…ecuación 88

$Ancho\ interno\ 1 = 35\ mm$…ecuación 89

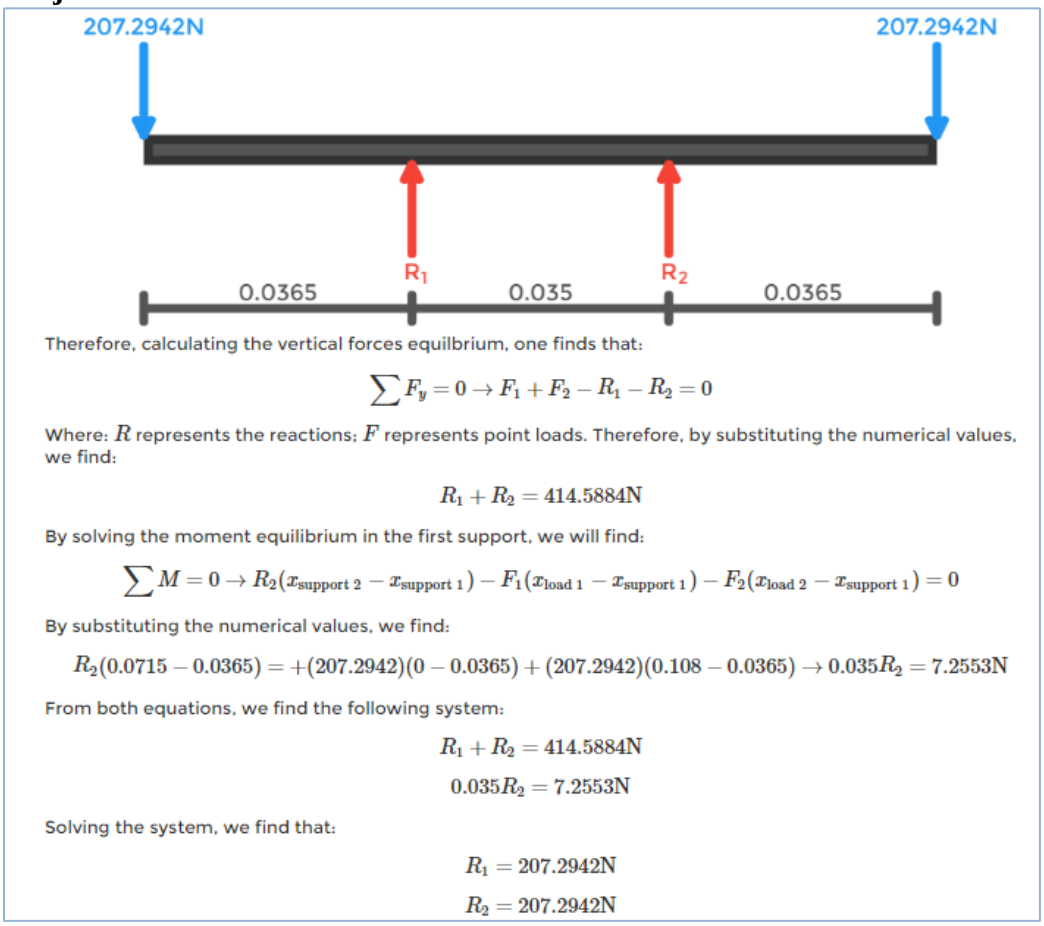
$Ancho\ interno\ 2 = 50\ mm$…ecuación 90

$F_{maxVertical} = 490.5 \cos 25 = 444.5439\ N$…ecuación 91

$F_{maxHorizontal} = 490.5 \sin 25 = 207.2942\ N$…ecuación 92

$Ancho\ total\ buje = 108\ mm$…ecuación 93

Cálculo del buje con ancho interno de 35 mm usando VIGA Online:



Therefore, calculating the vertical forces equilbrium, one finds that:

$$\sum F_y = 0 \rightarrow F_1 + F_2 - R_1 - R_2 = 0$$

Where: R represents the reactions; F represents point loads. Therefore, by substituting the numerical values, we find:

$$R_1 + R_2 = 414.5884\text{N}$$

By solving the moment equilibrium in the first support, we will find:

$$\sum M = 0 \rightarrow R_2(x_{\text{support 2}} - x_{\text{support 1}}) - F_1(x_{\text{load 1}} - x_{\text{support 1}}) - F_2(x_{\text{load 2}} - x_{\text{support 1}}) = 0$$

By substituting the numerical values, we find:

$$R_2(0.0715 - 0.0365) = +(207.2942)(0 - 0.0365) + (207.2942)(0.108 - 0.0365) \rightarrow 0.035R_2 = 7.2553\text{N}$$

From both equations, we find the following system:

$$R_1 + R_2 = 414.5884\text{N}$$

$$0.035R_2 = 7.2553\text{N}$$

Solving the system, we find that:

$$R_1 = 207.2942\text{N}$$

$$R_2 = 207.2942\text{N}$$

Figura 40. Cálculo de las fuerzas de reacción horizontales causadas por las placas laterales sobre el buje con ancho de 35 mm.

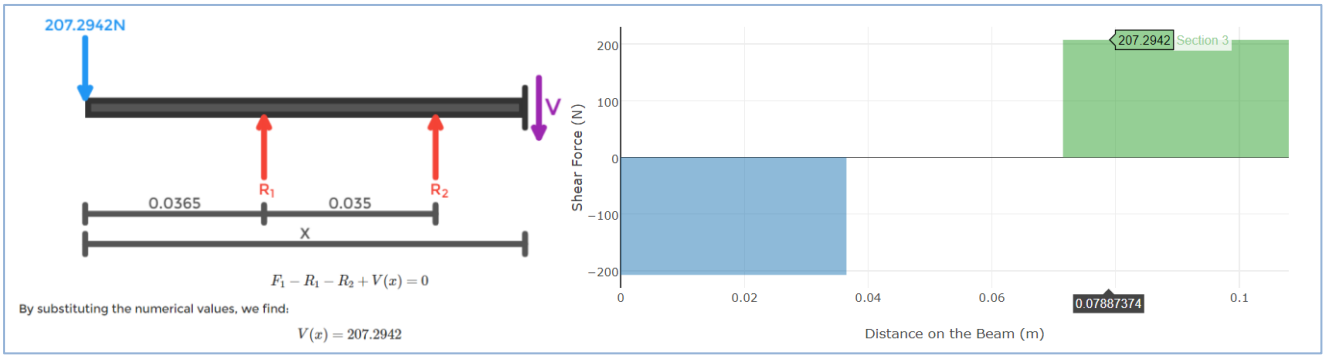


Figura 41. Diagrama de fuerza cortante - Reacción horizontal - Buje con ancho de 35 mm.

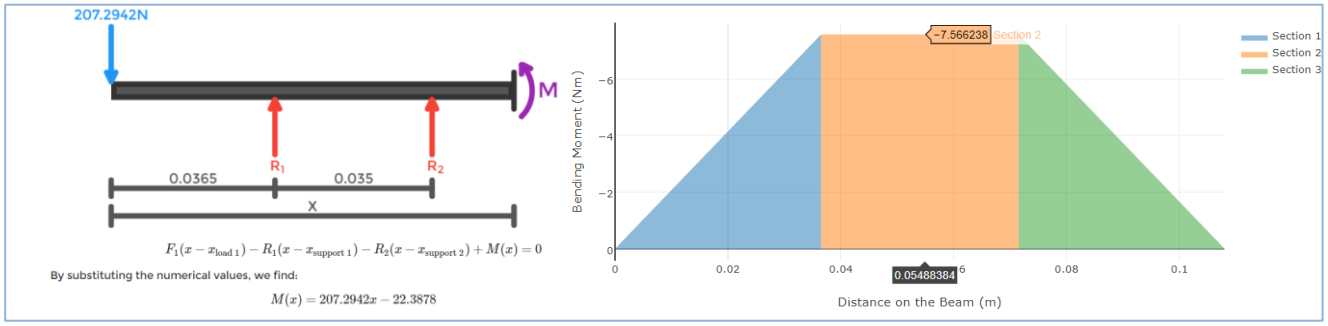


Figura 42. Diagrama de momento flexionante - Reacción horizontal - Buje con ancho de 35 mm.

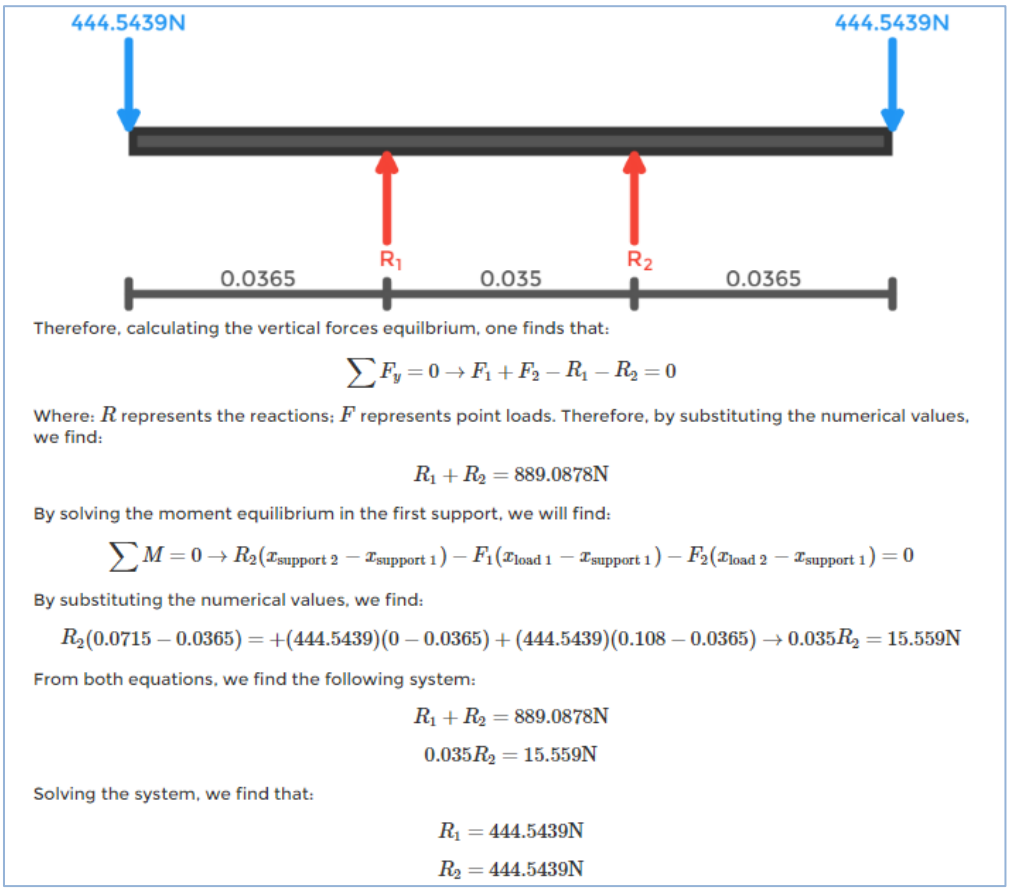


Figura 43. Cálculo de las fuerzas de reacción verticales causadas por las placas laterales sobre el buje con ancho de 35 mm.

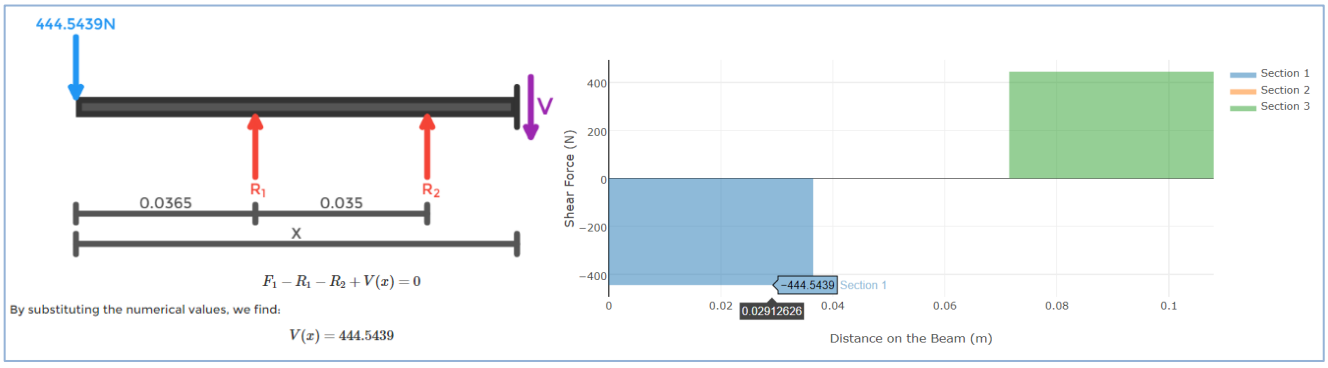


Figura 44. Diagrama de fuerza cortante - Reacción vertical - Buje con ancho de 35 mm.

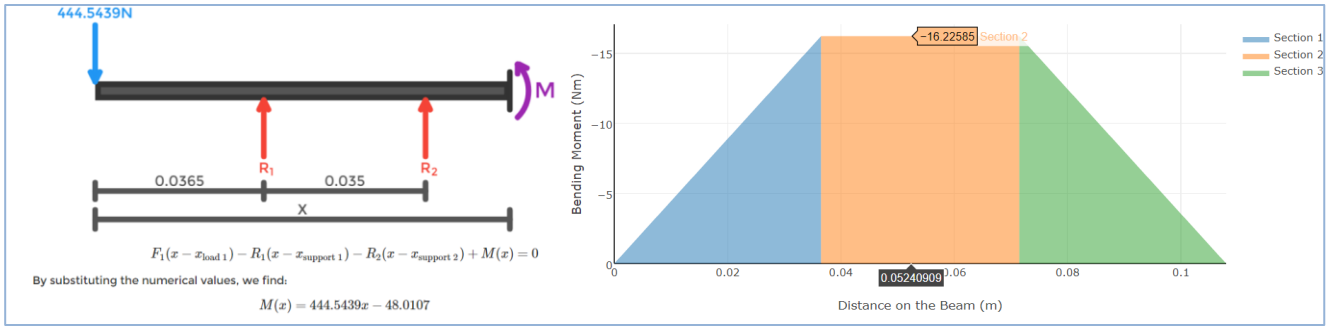


Figura 45. Diagrama de momento flexionante - Reacción vertical - Buje con ancho de 35 mm.

$$Mmax_{Horizontal} = 7.5662\ N * m \ldots \text{ecuación 94}$$

$$Mmax_{Vertical} = 16.2258\ N * m \ldots \text{ecuación 95}$$

Cálculo del buje con ancho interno de 50 mm usando VIGA Online:

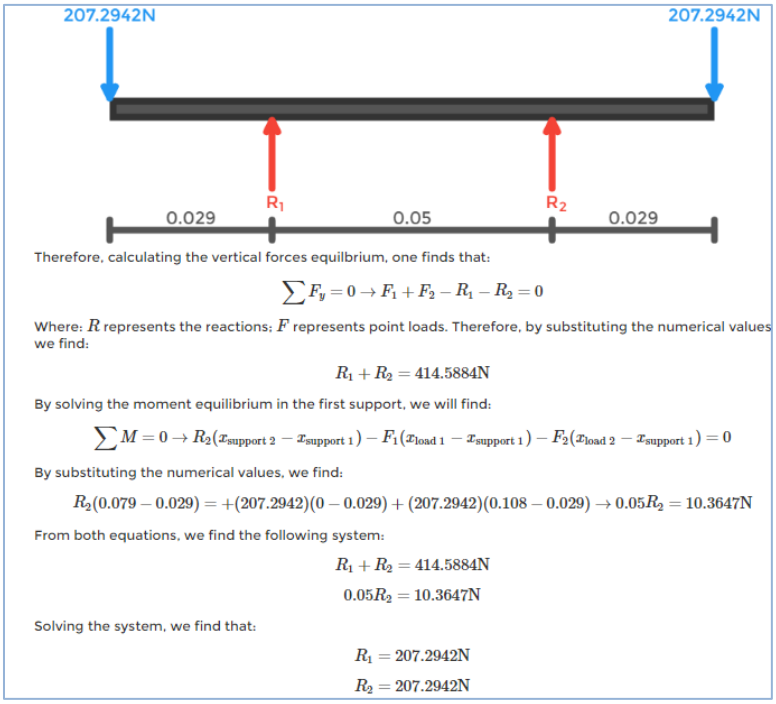


Figura 46. Cálculo de las fuerzas de reacción horizontales causadas por las placas laterales sobre el buje con ancho de 50 mm.

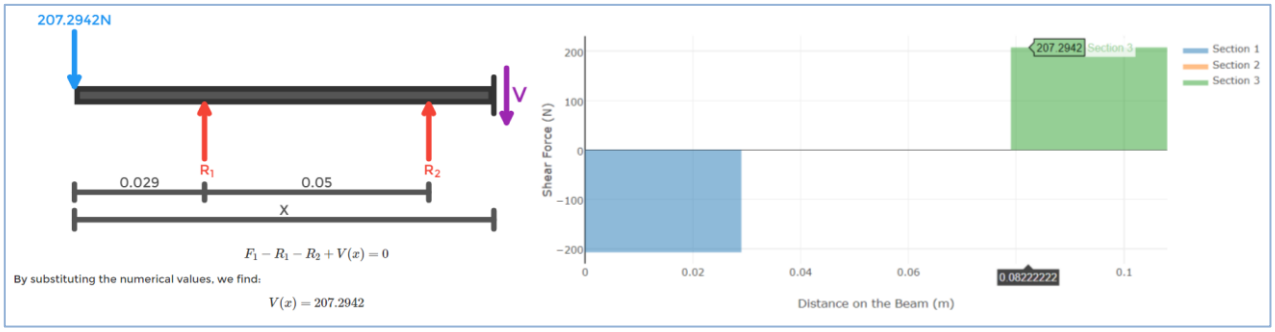


Figura 47. Diagrama de fuerza cortante - Reacción horizontal - Buje con ancho de 50 mm.

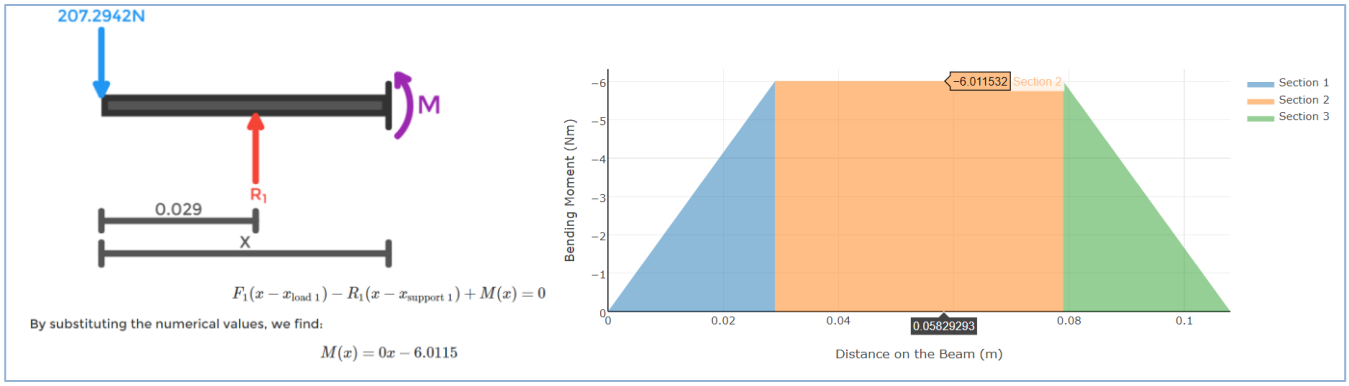


Figura 48. Diagrama de momento flexionante - Reacción horizontal - Buje con ancho de 50 mm.

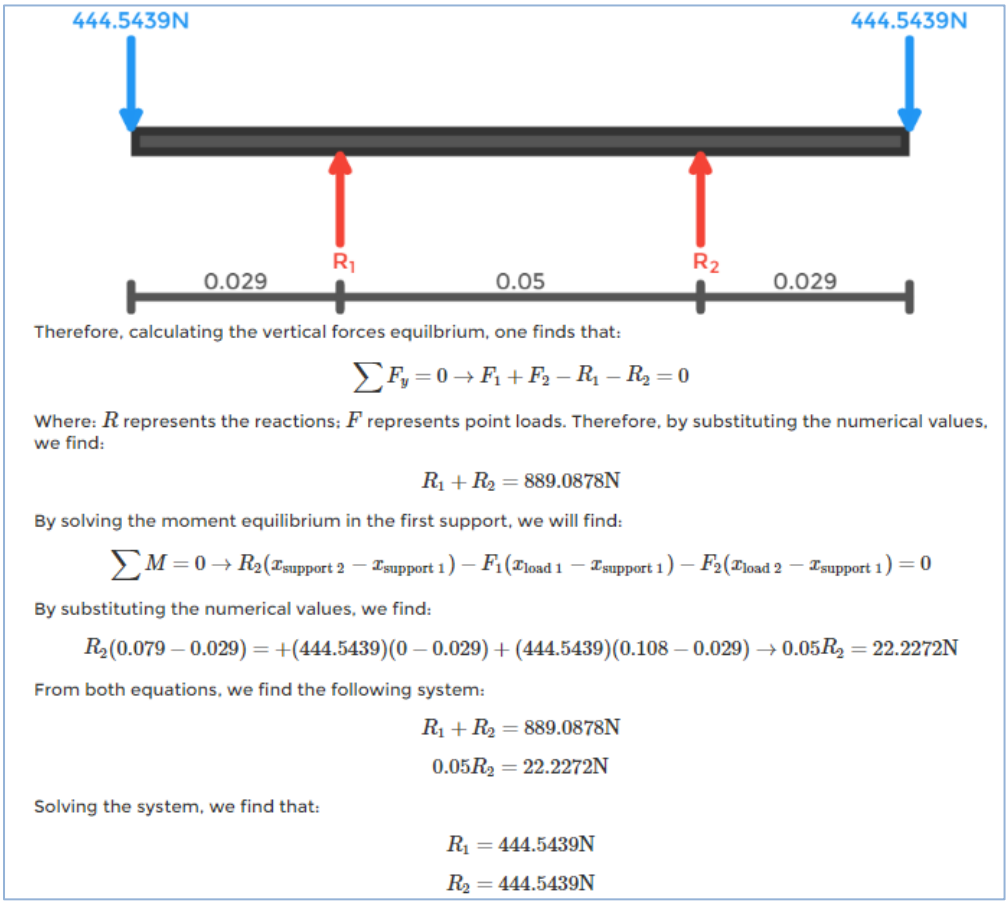


Figura 49. Cálculo de las fuerzas de reacción verticales causadas por las placas laterales sobre el buje con ancho de 50 mm.

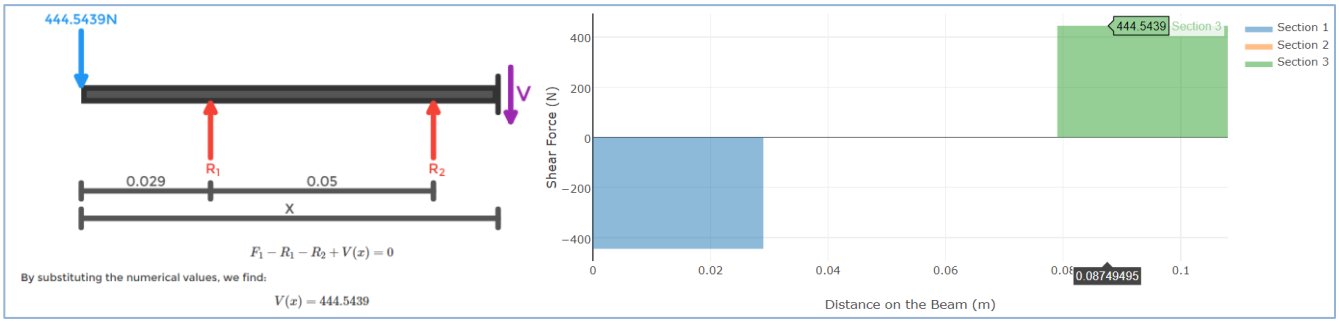


Figura 50. Diagrama de fuerza cortante - Reacción vertical - Buje con ancho de 50 mm.

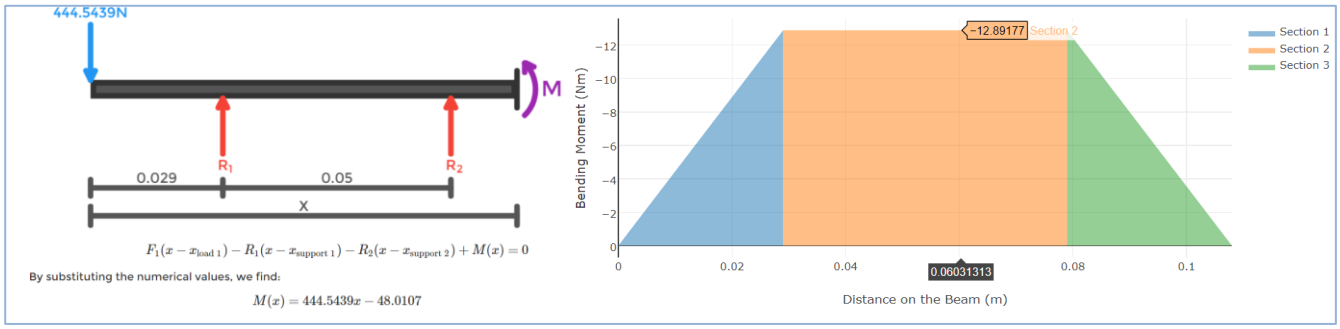


Figura 51. Diagrama de momento flexionante - Reacción vertical - Buje con ancho de 50 mm.

$$Mmax_{Horizontal} = 6.0115\ N * m \ldots \text{ecuación 96}$$

$$Mmax_{Vertical} = 12.8917\ N * m \ldots \text{ecuación 97}$$

Cuando en una viga se están aplicando dos fuerzas en el mismo punto, pero en distinto plano, lo que se debe hacer para obtener el momento flexionante total es sumar sus cuadrados y sacar la raíz del resultado para que en base a ese momento se obtenga el esfuerzo del elemento circular hueco causado por flexión:

$$Mmax = \sqrt{Mmax_{Horizontal}{}^2 + Mmax_{HorizontalVertical}{}^2}$$

…ecuación 98

$$\sigma_F = \frac{32(Mmax)}{\pi {d_e}^3 \left(1 - \left(\frac{d_i}{d_e}\right)^4\right)}$$

…ecuación 99

Cálculo de esfuerzo por flexión en buje con ancho de 35 mm:

$$Mmax = \sqrt{7.5662^2 + 16.2258^2} = 17.9031$$

…ecuación 100

$$\sigma_F = \frac{32(17.9031)}{\pi(0.012^3)\left(1 - \left(\frac{0.005}{0.012}\right)^4\right)} = 108.8117\ MPa$$

…ecuación 101

Cálculo de esfuerzo por flexión en buje con ancho de 50 mm:

$$Mmax = \sqrt{6.0115^2 + 12.8917^2} = 14.2244$$

…ecuación 102

$$\sigma_F = \frac{32(14.2244)}{\pi(0.012^3)\left(1 - \left(\frac{0.005}{0.012}\right)^4\right)} = 86.4533\ MPa$$

…ecuación 103

En base a los resultados obtenidos para ambos tipos de buje, se puede concluir que la barra circular del catálogo de Metales Díaz mostrado en el **Apéndice C** que es de aluminio aleación 6061-T6 y tiene un esfuerzo de cedencia de 240 MPa, soporta el esfuerzo de trabajo al aplicar el peso máximo del usuario sin aumentar mucho el peso de la estructura, por lo que se escoge como material para el buje y el diámetro elegido para él es de 44.4 mm [37] ya que se piensa unir el eje con la placa por medio de 6 remaches con diámetro de 5 mm.

c.2. Sistema de captación y acondicionamiento de energía eléctrica

Como primer paso para acondicionar la señal, las salidas del motor sin escobillas (motor trifásico) MXUS-XF07 se rectifican utilizando el rectificador trifásico de onda completa (figura 52). A continuación, se muestran los cálculos realizados para la rectificación.

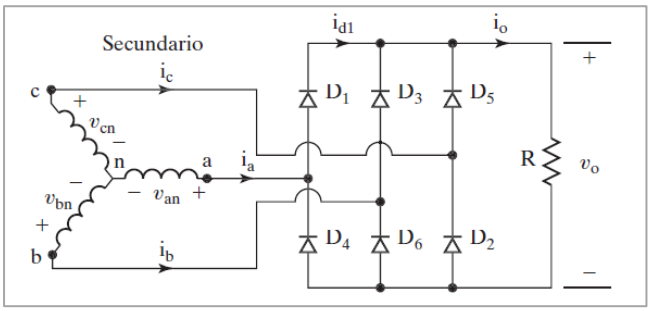


Figura 52. Rectificador trifásico de onda completa con puente de diodos.

Ecuaciones obtenidas de [41].

Datos:

$v_{cn} = 48\ V$…ecuación 104
$60\ Hz$
$0°$

$v_{an} = 48\ V$…ecuación 105
$60\ Hz$
$-120°$

$v_{bn} = 48\ V$…ecuación 106
$60\ Hz$
$120°$

$R_L = 10\ \Omega$…ecuación 107

$V_{rms} = v_{an} = v_{bn} = v_{cn}$…ecuación 108
$V_m = \sqrt{2} V_{rms}$…ecuación 109
$V_m = \sqrt{2} * 48\ V = 67.88\ V$…ecuación 110

Cálculos:

Tensión CD en la carga

$V_{cd\ RL} = \frac{3\sqrt{3}}{\pi} V_m$…ecuación 111

$V_{cd\ RL} = \frac{3\sqrt{3}}{\pi} * 67.88\ V = 112.27\ V$…ecuación 112

Corriente CD en la carga

$I_{cd\ RL} = \frac{V_{cd\ RL}}{R_L}$…ecuación 113

$I_{cd\ RL} = \frac{112.27\ V}{10\ \Omega} = 11.22\ A$…ecuación 114

Tensión RMS en la carga

$V_{rms\ RL} = \left(\frac{3}{2} + \frac{9\sqrt{3}}{4\pi}\right)^{1/2} * V_m$…ecuación 115

$V_{rms\ RL} = \left(\frac{3}{2} + \frac{9\sqrt{3}}{4\pi}\right)^{1/2} * 67.88\ V = 112.37\ V$
…ecuación 116

Corriente RMS en la carga

$I_{rms\ RL} = \frac{V_{rms\ RL}}{R_L}$…ecuación 117

$I_{rms\ RL} = \frac{112.37\ V}{10\ \Omega} = 11.23\ A$…ecuación 118

Potencias de salida

$P_{cd\ salida} = V_{cd\ RL} * I_{cd\ RL}$…ecuación 119
$P_{cd\ salida} = 112.27\ V * 11.22\ A =$
$1259.66\ W$…ecuación 120

$P_{ca\ salida} = V_{rms\ RL} * I_{rms\ RL}$…ecuación 121
$P_{ca\ salida} = 112.37\ V * 11.23\ A = 1261.91\ W$
…ecuación 122

Eficiencia del rectificador

$\eta = \frac{P_{cd\ salida}}{P_{ca\ salida}} * 100$…ecuación 123

$\eta = \frac{1259.66\ W}{1261.91} * 100 = 99.82\ \%$…ecuación 124

Factor de forma

$F.F = \frac{V_{rms\ RL}}{V_{cd\ RL}}$…ecuación 125

$F.F = \frac{112.37\ V}{112.27} = 1.00089$…ecuación 126

Factor de componente ondulatoria

$R.F = \sqrt{(F.F)^2 - 1}$…ecuación 127

$R.F = \sqrt{(1.00089)^2 - 1} = 0.0422$…ecuación 128

Corriente pico a través de un diodo

$I_m = \frac{\sqrt{3}V_m}{R_L}$…ecuación 129

$I_m = \frac{\sqrt{3}*67.88\ V}{10\ \Omega} = 11.75\ A$…ecuación 130

Valor RMS de la corriente en cada diodo

$I_{Drms} = 0.5518\ I_m$…ecuación 131

$I_{Drms} = 0.5518 * 11.75\ A = 6.48\ A$…ecuación 132

Corriente en la fase

$I_s = I_m \left[\frac{2}{\pi}\left(\frac{\pi}{6} + \frac{1}{2} * \sin\frac{2\pi}{6}\right)\right]^{1/2}$ …ecuación 133

$I_s = 11.75\ A \left[\frac{2}{\pi}\left(\frac{\pi}{6} + \frac{1}{2} * \sin\frac{2\pi}{6}\right)\right]^{1/2} = 9.16\ A$

…ecuación 134

En la figura 1 del **Anexo 6** se muestran las formas de onda obtenidas al simular el circuito rectificador en Multisim.

La curva **verde** corresponde a Vcn con un desfase de 0°. La curva **roja** corresponde a Van con desfase de -120°. La curva **azul** corresponde a Vbn con un desfase de 120°.

En la figura 2 del **Anexo 6**, se muestra la curva **rosa** que corresponde a la señal rectificada en la resistencia de carga.

Para alimentar a una batería portable con 5V/2A o la batería de una bicicleta eléctrica con 54.6V/2A, se decidió utilizar un convertidor reductor/elevador (del inglés "*Buck-Boost"*), donde el voltaje de salida puede ser mayor o menor que el de la entrada, tomando en cuenta que la polaridad de la salida es negativa con respecto a la terminal común.

En el convertidor existen dos modos de operación principalmente por la corriente de salida, modo de conducción continua y modo conducción discontinuo. Se utilizó el modo de conducción discontinuo por que se encuentra más apegado a resultados reales. Para obtener el modo de conducción discontinuo se modificará el ciclo de trabajo de una señal PWM, para así conseguir las tensiones necesarias para alimentar las baterías.

Ecuaciones obtenidas de [40].

$V_o = 48\ V$…ecuación 135

$I_o = 0.5$ …ecuación 136

Tensión de entrada:

$V_d = 5\ V$…ecuación 137

Rizado de la corriente:

Corriente pico:

$I_{Lp} = \frac{V_d * D * T}{L}$…ecuación 150

$I_{Lp} = \frac{(5V)(0.8)(1/20kHz)}{1.481uH}$…ecuación 151

$I_{Lp} = 13.504\text{A}$…ecuación 152

$\frac{\Delta i_L}{I_L} = 5\%$ …ecuación 138

$f = 20kHz$…ecuación 139

Las potencias tanto de entrada como de salida deben ser iguales:

$P_d = P_o$…ecuación 140

Por lo tanto, las potencias de salida para 48V es:

$P_o = 48\,V * 0.5A = 27Watts$ …ecuación 141

A la entrada debe haber:

$P_d = 27Watts$…ecuación 142

Por lo cual se puede determinar su corriente de entrada:

$I_d = \frac{27Watts}{5V} = 5.4A$…ecuación 143

Se toma en cuenta un valor de ciclo de trabajo lógico para determinar la inductancia de un elemento del convertidor:

$D = 0.8$…ecuación 144

Por lo tanto:

$\Delta_1 = \frac{DV_d}{V_o} = \frac{(0.8)(5)}{54V} = 0.07$ …ecuación 145

Con la siguiente ecuación se puede obtener el valor del inductor:

$D = \frac{V_o}{V_d}\sqrt{\frac{2L}{R_L T}}$…ecuación 146

$L = \frac{{V_d}^2 * D^2 * R_L * T}{2 * {V_o}^2}$ …ecuación 147

$L = \frac{(5)^2(0.8)^2(108\Omega)(1/20kHz)}{(2)(54)^2}$ …ecuación 148

$L = 14.8uH$…ecuación 149

La corriente media de salida es de:

$I_{out} = \frac{(13.504\text{A})(0.07)}{2} = 0.472A$…ecuación 153

Para que el ciclo de trabajo dé una tensión de 5V, se realiza el siguiente cálculo:

$D = \frac{V_o}{V_d}\sqrt{\frac{2L}{R_L T}}$…ecuación 154

$$D = \frac{(5)}{(5)}\sqrt{\frac{2(14.8uH)}{(2.5\Omega)(1/20kHz)}} = 0.486$$

…ecuación 155

El valor 0.486 es un aproximado a 0.5 de ciclo de trabajo, el cual nos indica que es el mismo valor de entrada como de salida.

A continuación, en la figura 53, se muestra la simulación del circuito:

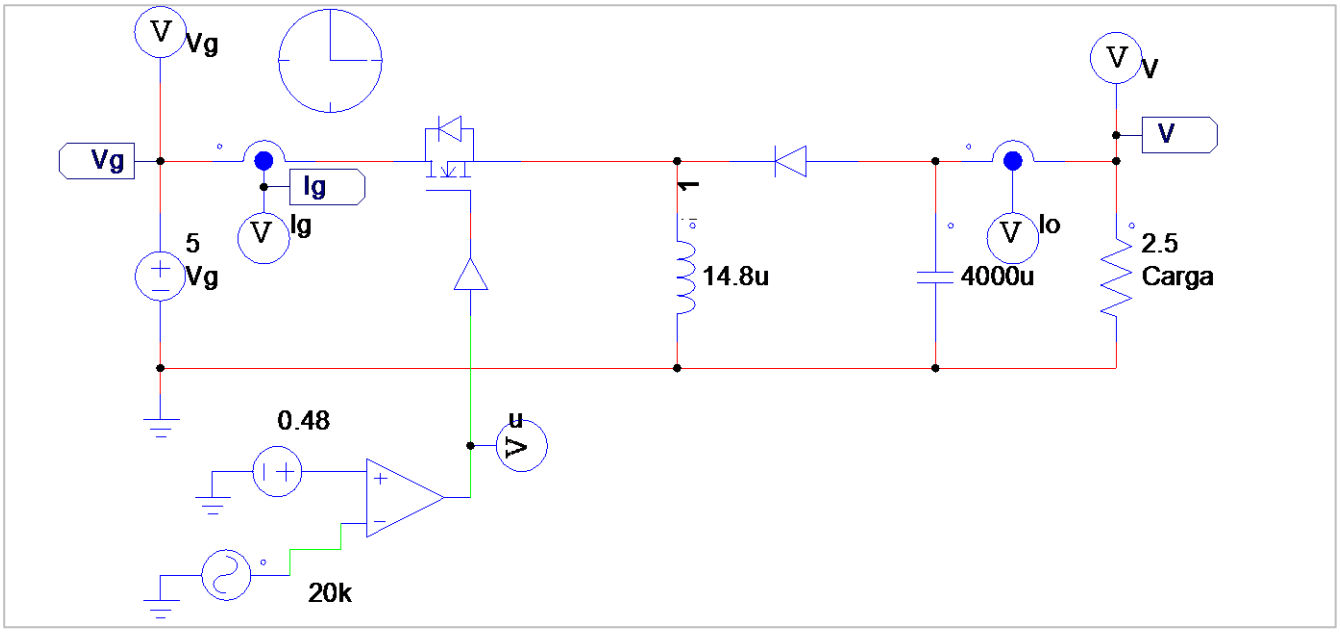


Figura 53. Circuito convertidor reductor/elevador simulado.

El resultado de la simulación es mostrado en la figura 54:

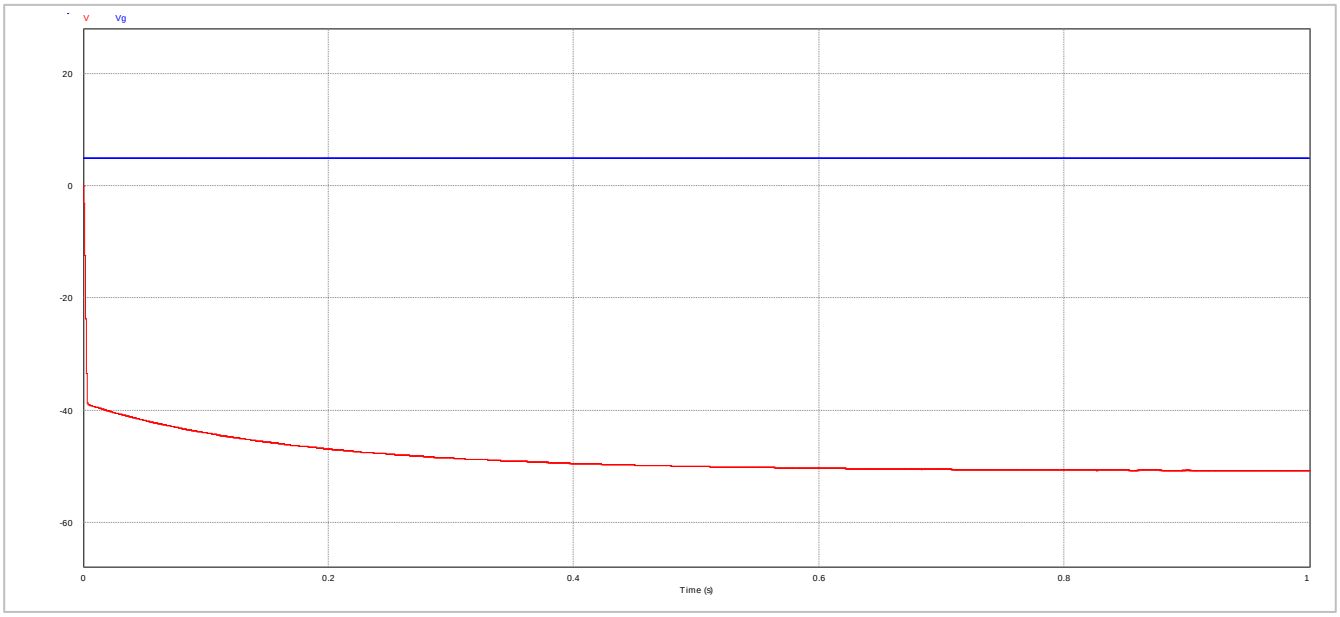


Figura 54. Resultado de la simulación.

Se observa que como salida de la gráfica de color rojo no está dando como resultado -50.6 V, respecto a tierra, por lo que la línea de color azul es la que se considera como la tensión de entrada del convertidor.

El circuito completo se muestra en la figura 55:

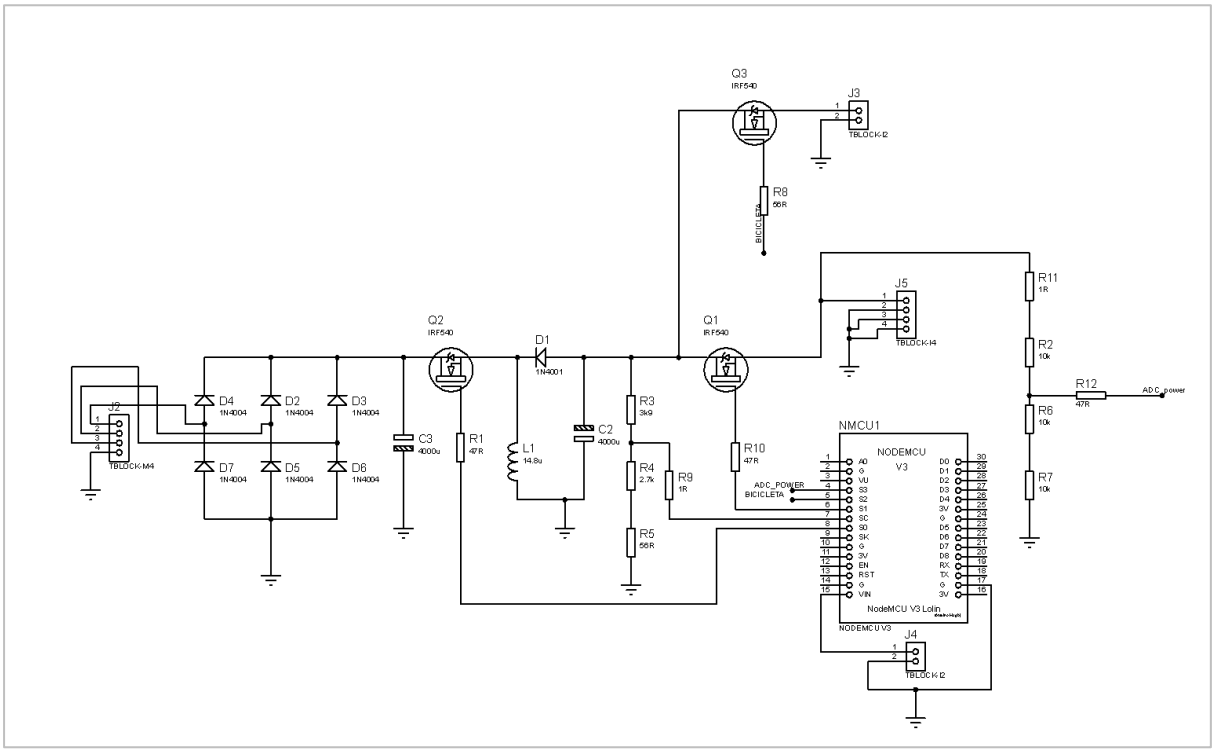


Figura 55. Circuito esquemático completo.

En la figura 56 se muestra el diseño del circuito impreso que representa el diagrama esquemático del circuito anterior:

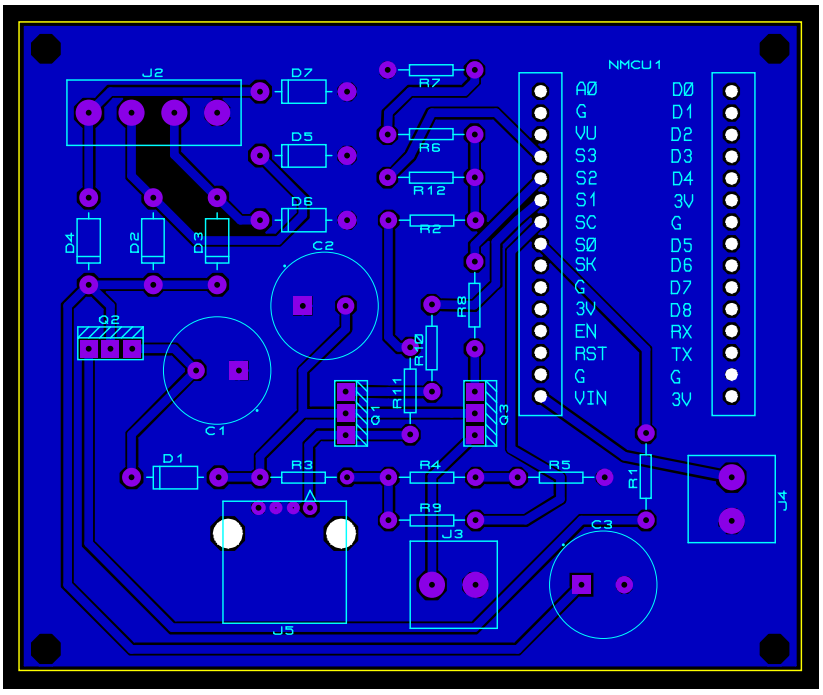


Figura 56. PCB del circuito.

En la figura 57 se muestra una vista previa en 3D de la placa que conforma el circuito:

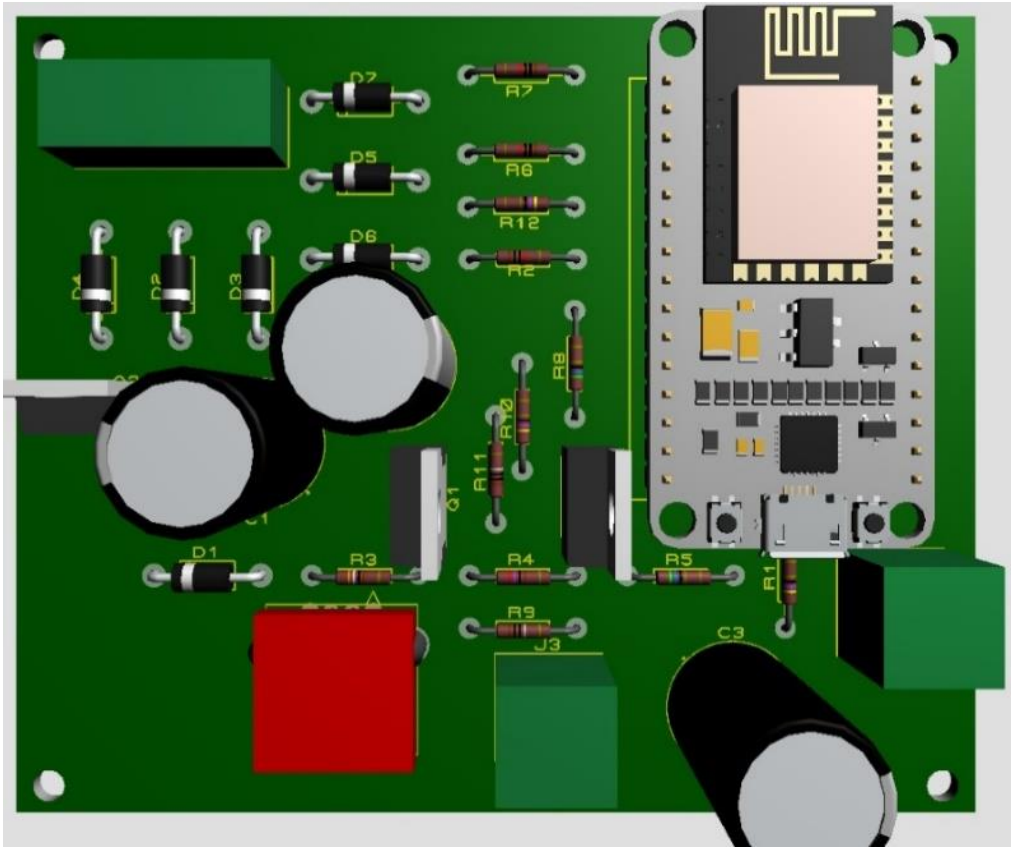


Figura 57. Vista previa en 3D del circuito impreso.

Para programar el microcontrolador ESP32, se desarrolló el código perteneciente al **Anexo 1** del presente documento, utilizando el lenguaje de programación Micropython.

d. Diseño detallado

Ya habiendo realizado el diseño de materialización se lleva a cabo el diseño detallado para ahondar en los resultados obtenidos y así obtener un diseño robusto del sistema.

d.1. Estructura mecánica

Ya habiendo calculado de forma analítica el esfuerzo mecánico del eje con un ancho de 35 y 50 mm respectivamente, tomando en cuenta solo su diámetro exterior e interior, es momento de encontrar el redondeo óptimo del buje real, ya que como se piensa unir las placas laterales con el buje por medio de 6 remaches para crear la estructura mecánica, habrá un cambio de diámetro, lo que generará una esquina en donde se puede crear un concentrador de esfuerzo, por lo que se modelará el buje con la carga del peso del usuario de 100 kg en COMSOL y se iterará con redondeos partiendo de 0.5 mm hasta que se encuentre el óptimo, para ello se tomará como referencia el eje con ancho de 50 mm que tenía como resultado analítico un esfuerzo de 86.4533 MPa, por lo que el esfuerzo de las iteraciones no se debe alejar mucho de este valor (figuras 58 a 63).

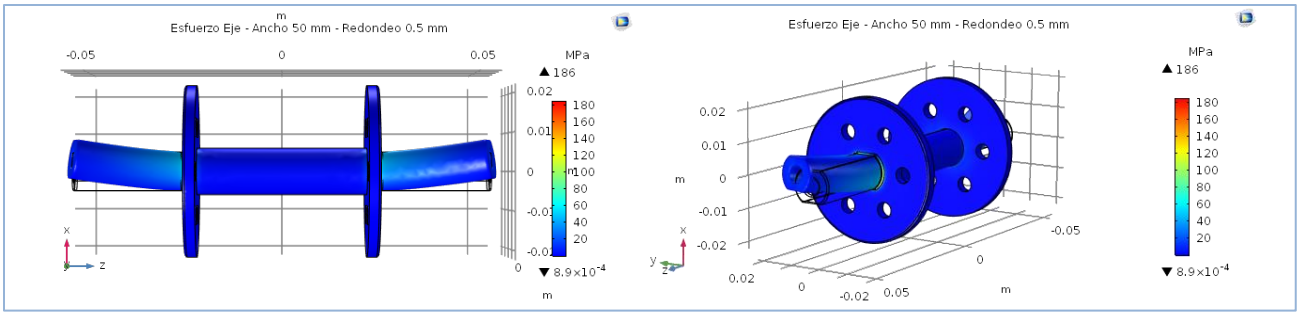


Figura 58. Concentrador de esfuerzo del buje con ancho de 50 mm y redondeo de 0.5 mm – Esfuerzo máximo: 186 MPa.

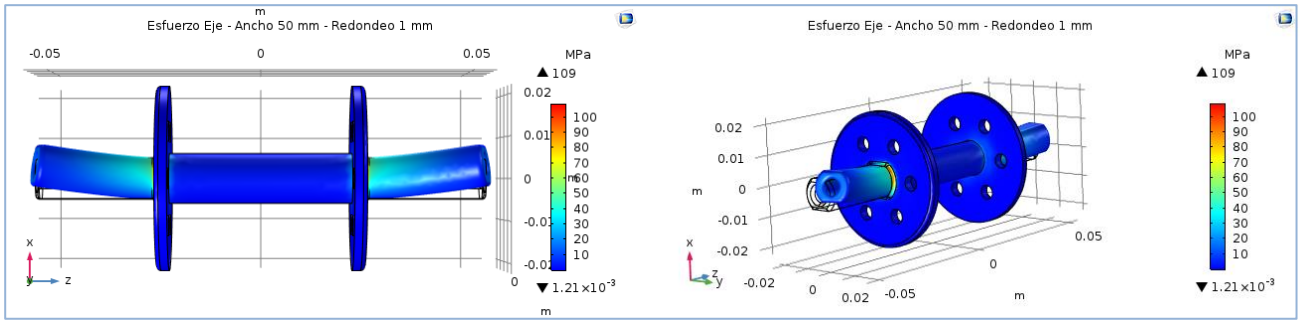


Figura 59. Concentrador de esfuerzo del buje con ancho de 50 mm y redondeo de 1 mm – Esfuerzo máximo: 109 MPa.

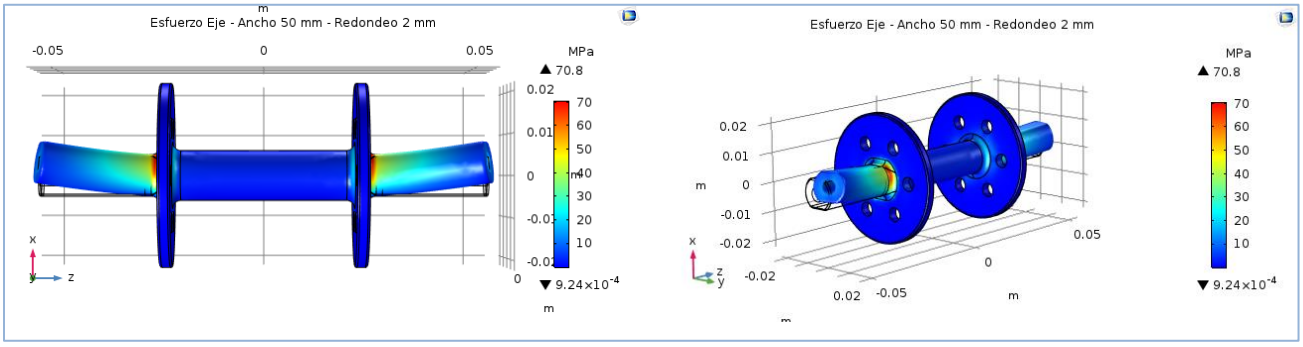


Figura 60. Concentrador de esfuerzo del buje con ancho de 50 mm y redondeo de 2 mm – Esfuerzo máximo: 70.8 MPa.

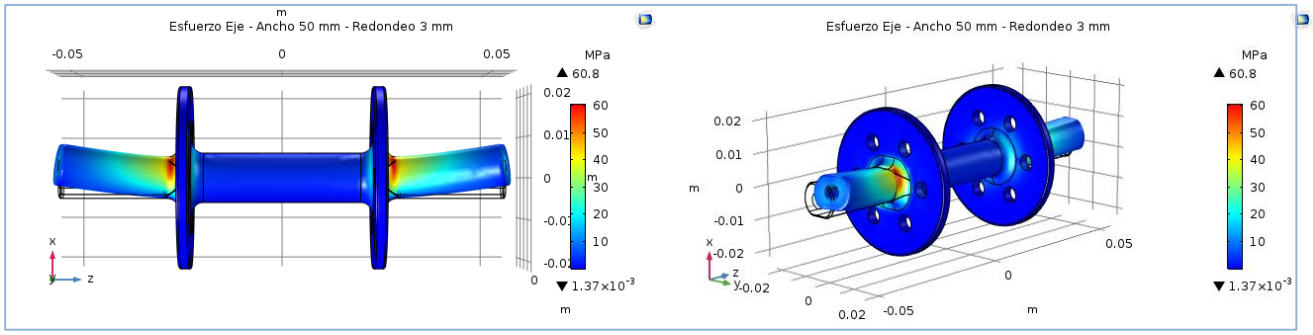


Figura 61. Concentrador de esfuerzo del buje con ancho de 50 mm y redondeo de 3 mm – Esfuerzo máximo: 60.8 MPa.

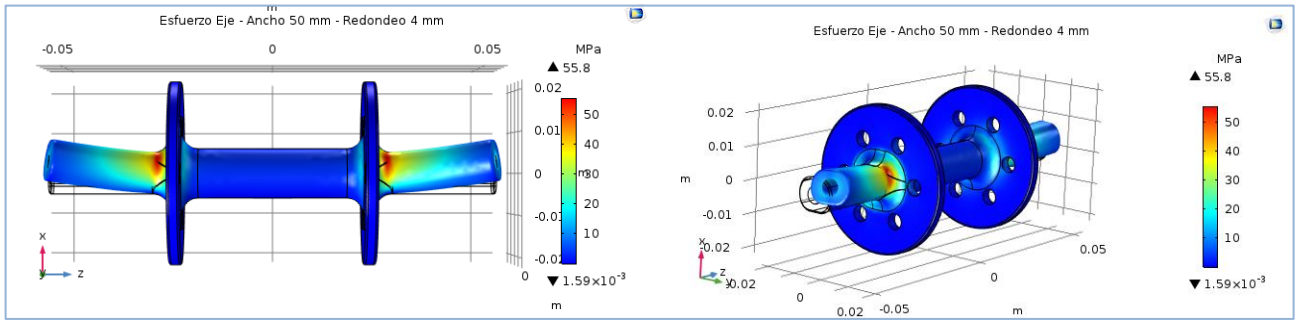


Figura 62. Concentrador de esfuerzo del buje con ancho de 50 mm y redondeo de 4 mm – Esfuerzo máximo: 55.8 MPa.

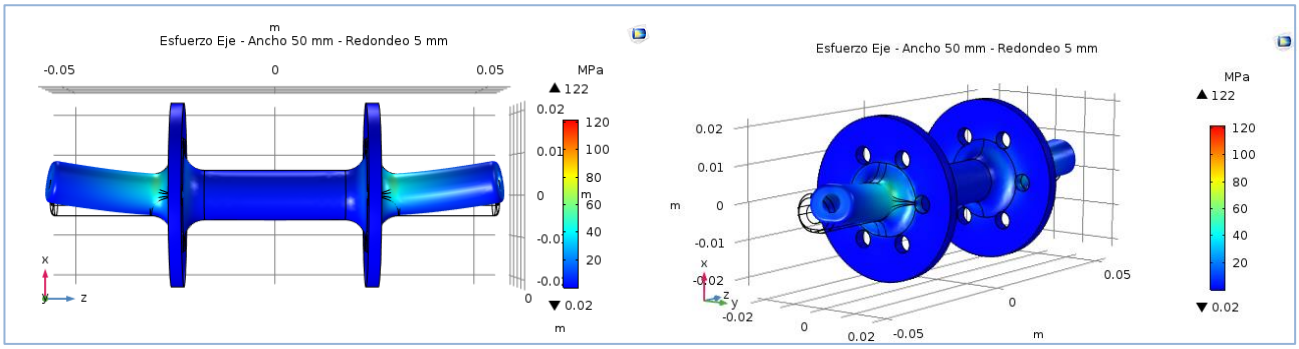


Figura 63. Concentrador de esfuerzo del buje con ancho de 50 mm y redondeo de 5 mm – Esfuerzo máximo: 122 MPa.

Por lo tanto, se puede concluir que el redondeo óptimo para el buje es de 4 mm, por lo que ahora se hará una prueba con el ancho de 35 y 50 mm para comparar resultados (figuras 64 y 65).

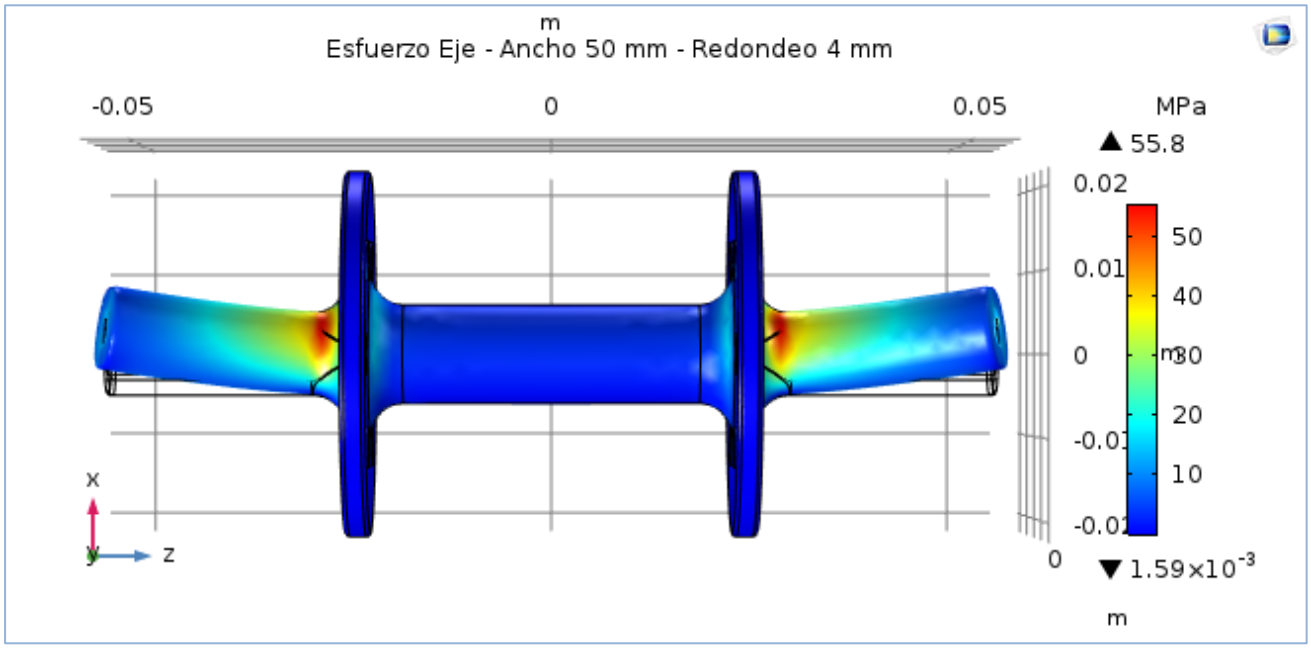


Figura 64. Concentrador de esfuerzo con redondeo óptimo de 4 mm – Ancho de 50 mm - Esfuerzo máximo: 55.8 MPa.

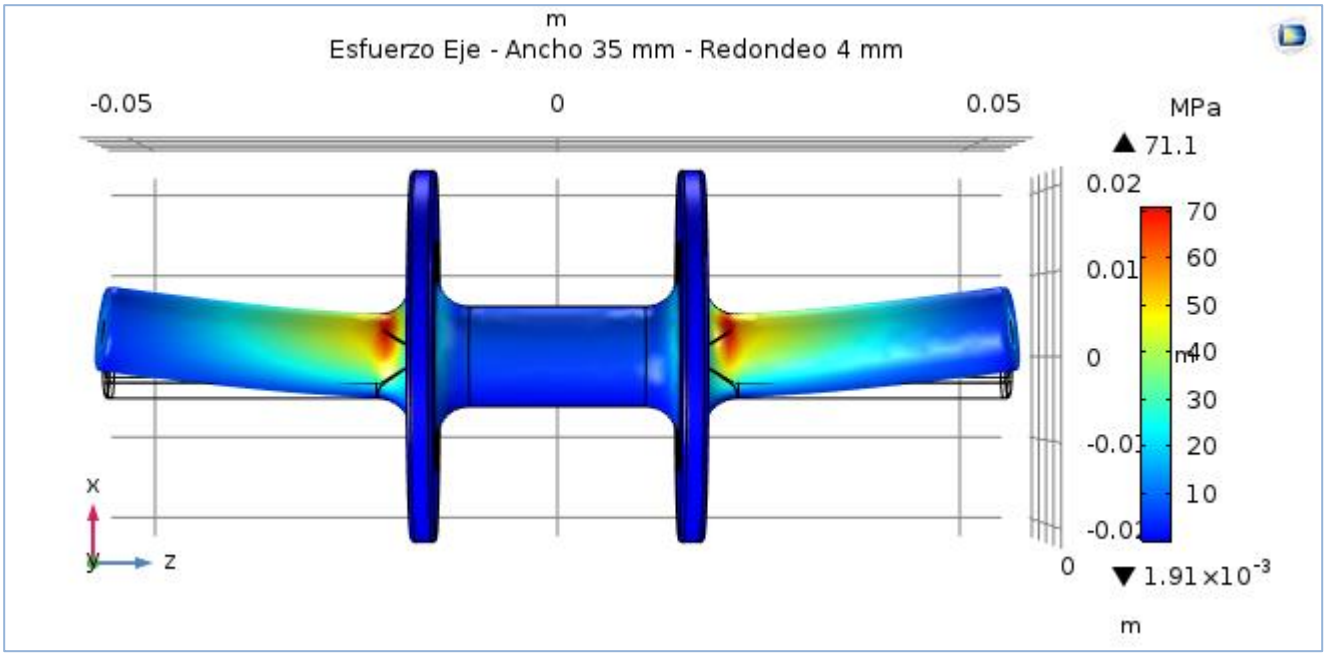


Figura 65. Concentrador de esfuerzo con redondeo óptimo de 4 mm – Ancho de 35 mm – Esfuerzo máximo: 71.1 MPa.

Tomando en cuenta que el esfuerzo de cedencia del Aluminio 6061-T6 es de 240 MPa, los factores de seguridad para los dos ejes son los siguientes:

Factor de seguridad del buje redondeo de 4 mm y ancho de 35 mm:

$\sigma max = 71.1\ MPa$…ecuación 156

$F.S. = \frac{\sigma_{pc}}{\sigma_t} = \frac{240}{71.1} = 3.3755$…ecuación 157

Factor de seguridad del buje redondeo de 4 mm y ancho de 50 mm:

$\sigma max = 55.8\ MPa$…ecuación 158

$F.S. = \frac{\sigma_{pc}}{\sigma_t} = \frac{240}{55.8} = 4.3010$…ecuación 159

También se realizaron pruebas de esfuerzo a la placa lateral de la estructura mecánica, se realizaron directamente a través de software debido a la forma inusual de los concentradores de esfuerzo que conforman la figura de la placa lateral en la estructura mecánica, para ello se aplicaron todas las fuerzas externas, como el peso de los motores, las fuerzas de reacción aplicadas por el buje sobre la placa, la fuerza lateral de reacción en la rueda tipo polea como respuesta a la componente horizontal de la carga impuesta por el usuario sobre la horquilla y por último la fuerza normal. Además, se fijaron las partes de la placa que se mantendrán estáticas durante la rotación de la rueda como el eslabón inferior, superior y los brazos laterales que estarán sujetos a las ruedas tipo polea. Estas fuerzas se aplicaron a la placa, iterando con sus distintos espesores para denotar cuál es el mejor calibre para la estructura mecánica (figuras 66, 67 y 68).

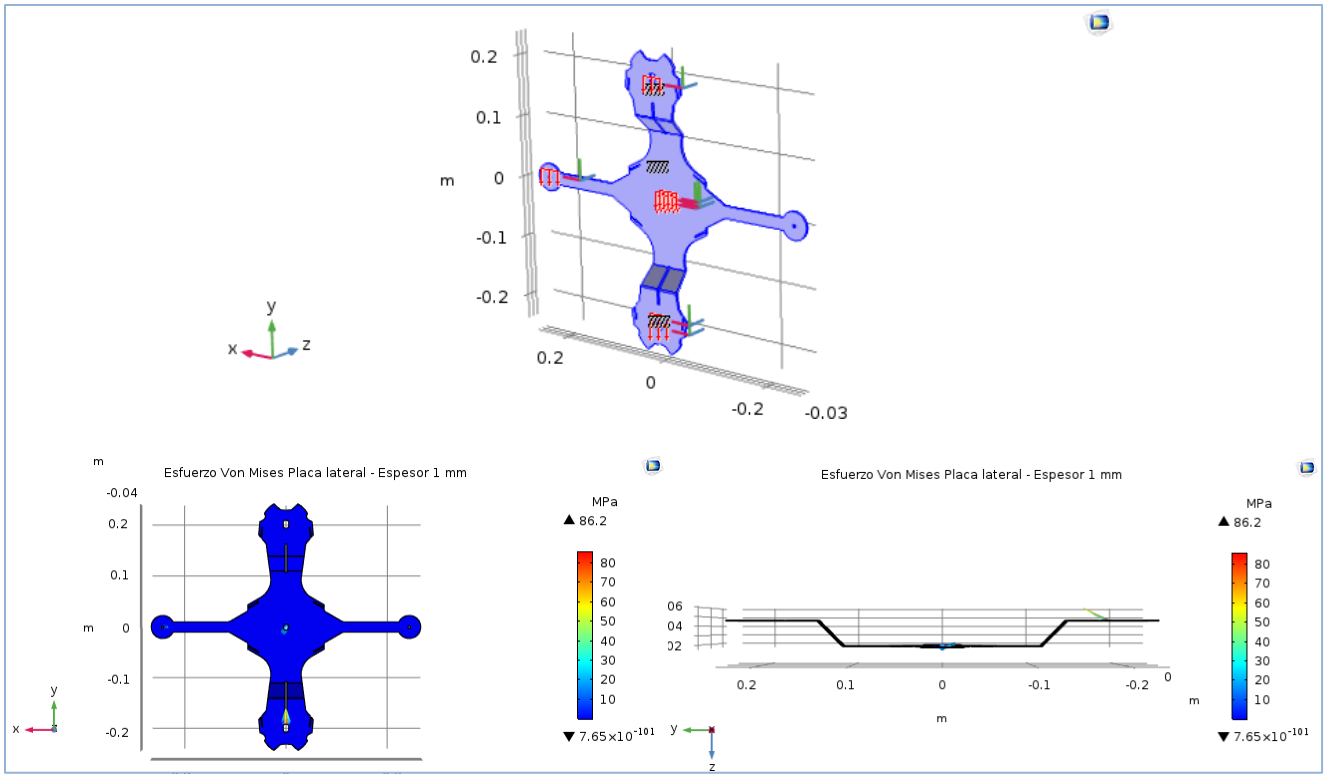


Figura 66. Fuerzas externas y Esfuerzo de la placa lateral con espesor de 1 mm – Esfuerzo máximo: 86.2 MPa.

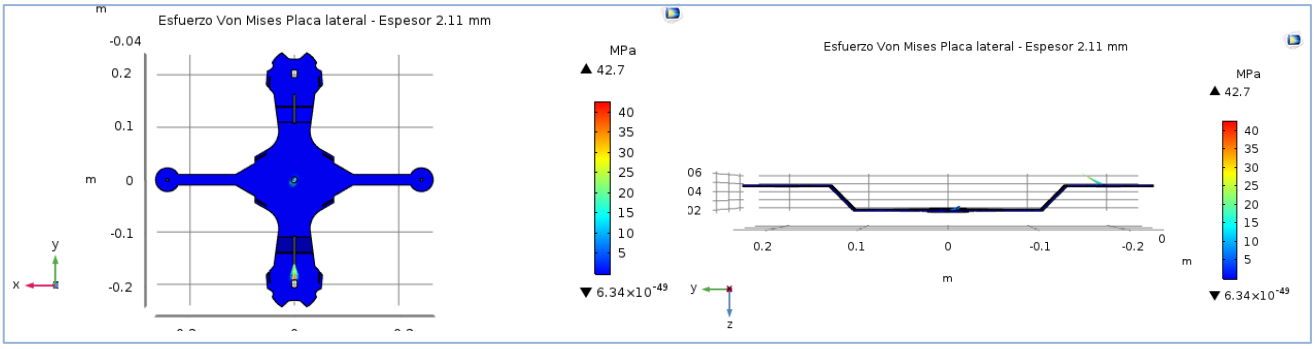


Figura 67. Esfuerzo de la placa lateral con espesor de 2.11 mm – Esfuerzo máximo: 42.7 MPa.

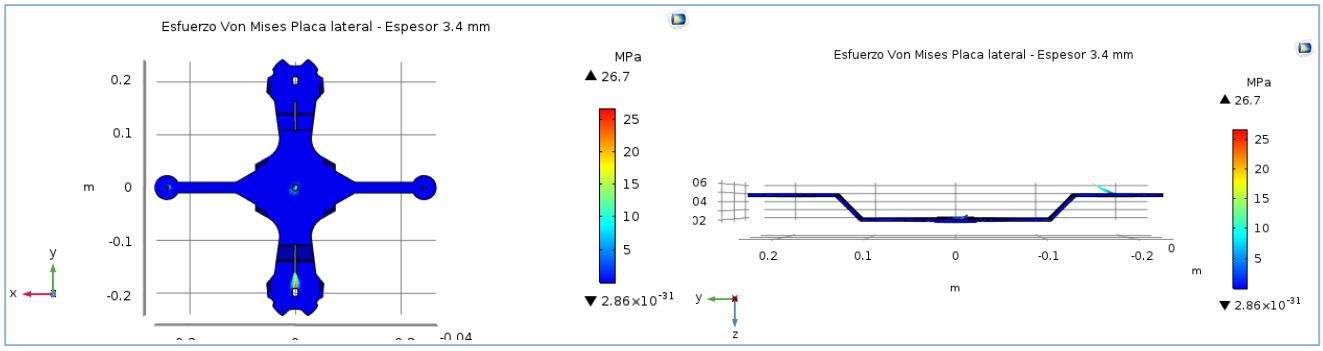


Figura 68. Esfuerzo de la placa lateral con espesor de 3.4 mm – Esfuerzo máximo: 26.7 MPa.

Se puede observar en la figura 68 que la opción obvia es la placa lateral con espesor de 3.4 mm debido a su muy bajo esfuerzo máximo, por lo que es el espesor óptimo para las dos placas laterales del ensamble con el aluminio 3003-H14.

Factor de seguridad de la placa lateral con el aluminio 3003-H14:

$\sigma max = 26.7\ MPa$…ecuación 160

$F.S. = \frac{\sigma_{pc}}{\sigma_t} = \frac{82.7371}{26.7} = 3.0987$…ecuación 162

Factor de seguridad de la placa lateral con el aluminio 1100-H14:

$\sigma max = 26.7\ MPa$…ecuación 161

$F.S. = \frac{\sigma_{pc}}{\sigma_t} = \frac{68.9476}{55.8} = 2.5823$…ecuación 163

Como la estructura es de aluminio para reducir lo más que se pueda el peso de la llanta frontal, se busca que la unión se haga por medio de remaches, el remache comercial más largo encontrado tiene 30 mm de profundidad y como las placas laterales tienen un espesor de 3.4 mm, el buje debe tener un ancho de 20.4 mm para dejar 3 mm de sobra y que la cabeza del remache pueda ejercer de forma correcta presión entre las dos placas laterales, por lo que se respeta el redondeo de 4 mm en los concentradores de esfuerzo y al ejecutar la simulación de esfuerzos con las mismas cargas podemos observar que el factor de seguridad sigue siendo aceptable, por lo que se opta finalmente por el buje con 20.2 mm de ancho (figura 69).

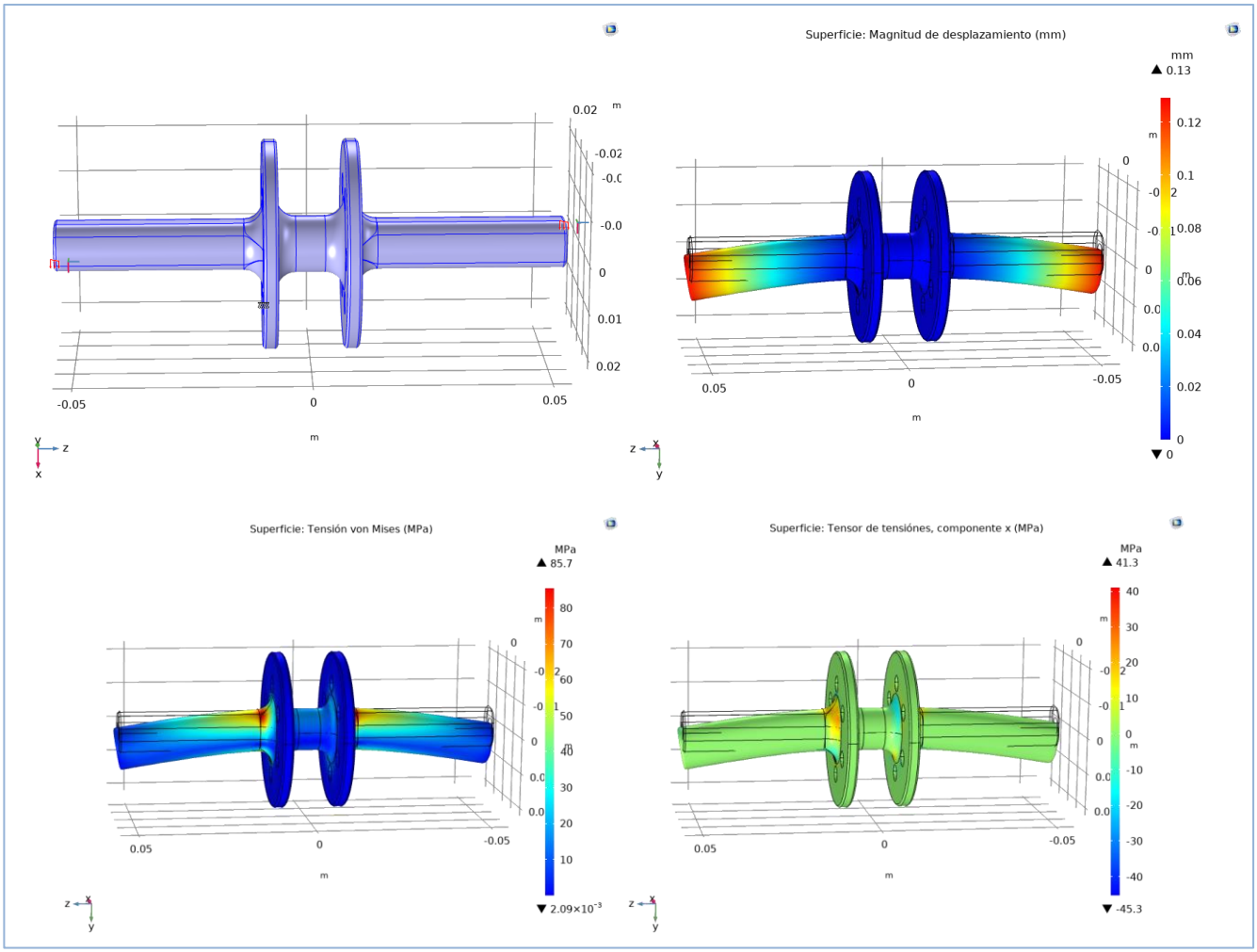


Figura 69. Concentrador de esfuerzo con redondeo óptimo de 4 mm – Ancho de 20.2 mm – Esfuerzo máximo: 85.7 MPa.

$$\sigma max = 85.7\ MPa \ldots \text{ecuación 164}$$

$$F.S. = \frac{\sigma_{pc}}{\sigma_t} = \frac{240}{85.7} = 2.8004 \ldots \text{ecuación 165}$$

Además, se debe considerar el espacio para colocar la batería portable y el sistema de captación y acondicionamiento de energía eléctica dentro de la estructura de la rueda frontal, por lo que una parte de las placas laterales debe doblarse para poder contener todo el sistema (figura 70).

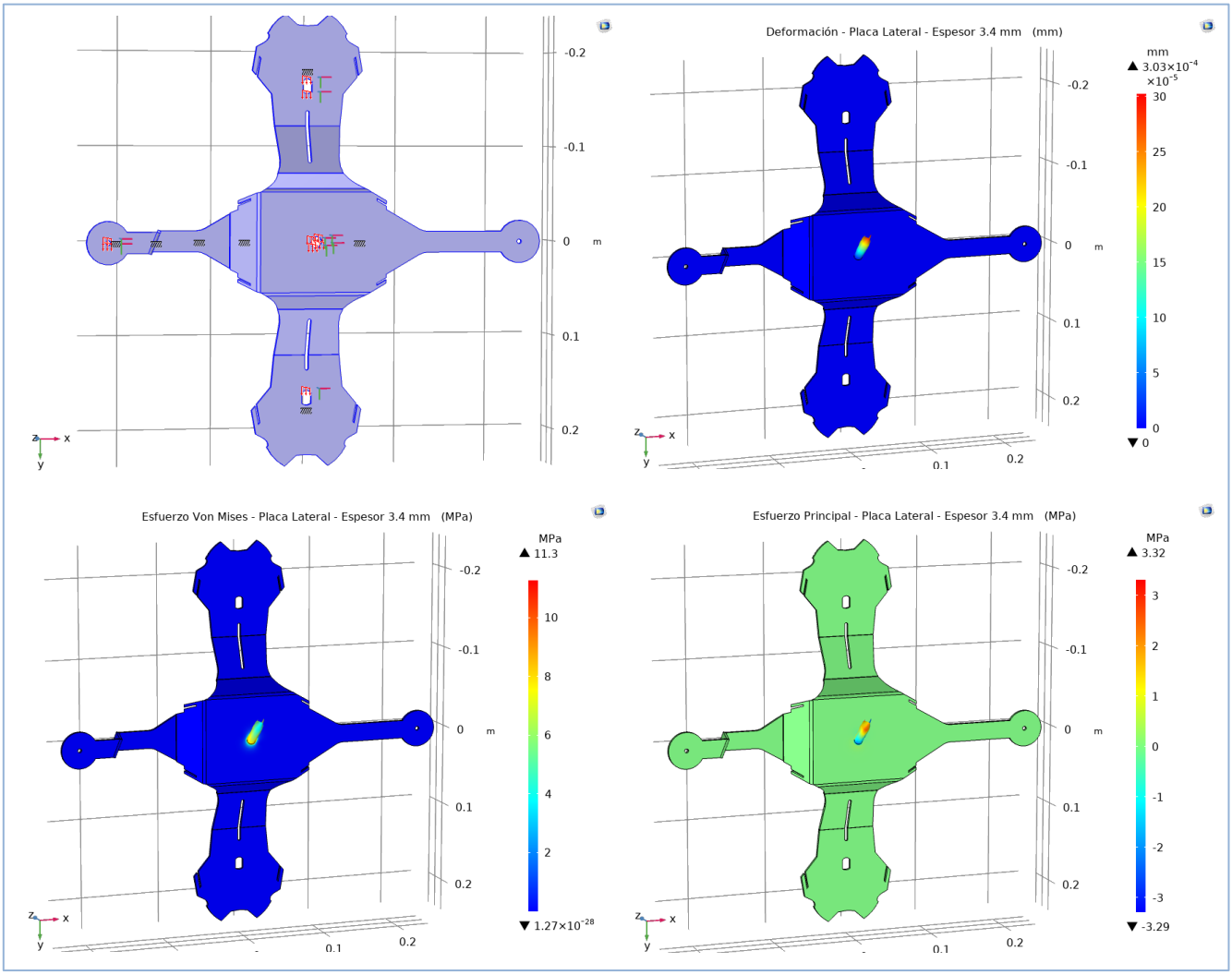


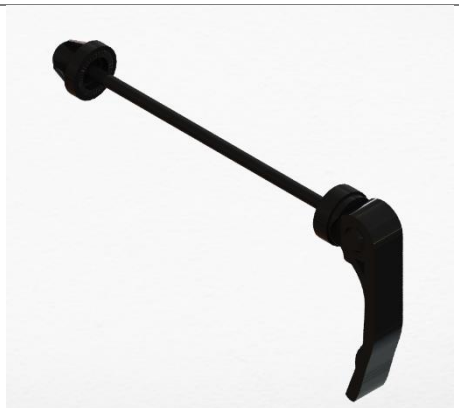
Figura 70. Esfuerzo de la placa lateral con espesor de 3.4 mm – Esfuerzo máximo: 11.3 MPa.

El ensamble de la estructura mecánica mostrado en la figura 71 se compone de varias piezas enlistadas a en la tabla 43, donde además se indica si la pieza será adquirida o manufacturada:

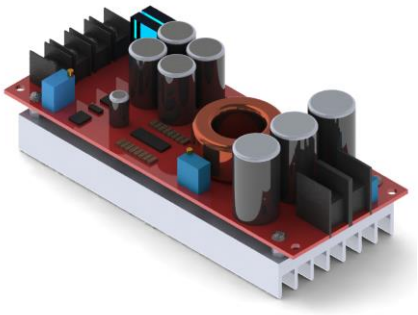
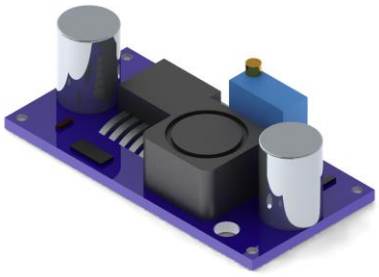
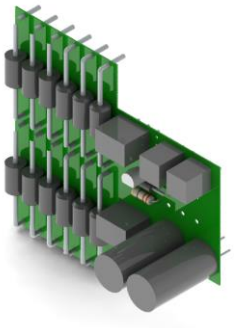
Figura 71. Ensamblaje de la rueda frontal de bicicleta generadora de energía eléctrica.

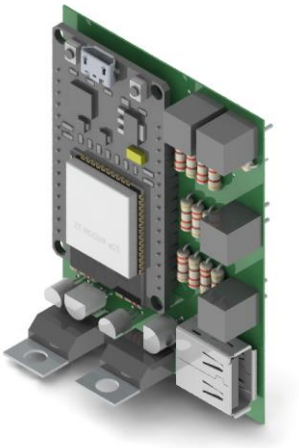
Tabla 43. Lista de piezas ensamblaje rueda frontal generadora de energía eléctrica.

Nombre y # de piezas en el ensamble	Pieza	Procedencia
1.- Llanta rodada 26 con rin tipo Endrick sin radios - 1 Pieza		Adquirida
2.- Generador eléctrico MXUS-XF07 - 2 Piezas		Adquirido

3.- Eje del buje de cierre rápido - 1 Pieza		Adquirido
4.- Bicicleta de pruebas sin rueda frontal - 1 Pieza		Adquirida
5.- Banda trapezoidal C64 - 1 Pieza		Adquirida
6.- Rueda lateral frontal tipo polea - 1 Pieza		Manufacturada

7.- Rueda lateral trasera tipo polea - 1 Pieza		Manufacturada
8.- Acoplamiento mecánico motor MXUS-XF07 y Banda trapezoidal C64 - 2 Piezas		Manufacturado
9.- Eje de rotación para el eje de cierre rápido - 1 Pieza		Manufacturado
10.- Placa lateral derecha de la estructura mecánica - 1 Pieza		Manufacturada

11.- Placa lateral izquierda de la estructura mecánica - 1 Pieza		Manufacturada
12.- Remaches Ø 4.6 X 30 (diámetro 4.6 mm, profundidad 30 mm) - 15 Piezas aproximadamente.		Adquiridos
13.- Convertidor elevador “Boost”.		Adquirido
14.- Convertidor reductor “Buck”		Adquirido
15.- Sistema de captación y acondicionamiento de energía eléctrica – PCB 1: Acondicionamiento de energía eléctrica.		Manufacturado

16.- Sistema de captación y acondicionamiento de energía eléctrica – PCB 2: Comunicación y control inalámbrica.		Manufacturado
17.- Dispositivo móvil o batería portable.		Adquirido

d.2. Eficiencia en la generación de energía eléctrica

Para calcular la eficiencia que teóricamente se obtendrá de la rueda frontal de bicicleta generadora de energía eléctrica se realiza el cálculo de la potencia mecánica que entra a la bicicleta de pruebas Monk Kron R26 18V y se compara con la potencia de salida obtenida en los motores sin escobillas MXUS-XF07 que se utilizarán como generadores.

d.2.1. Potencia mecánica de entrada

Se debe encontrar el coeficiente de fricción estático y dinámico entre los acoplamientos mecánicos hechos con filamento PLA y la banda trapezoidal C64 de caucho para comprobar si es fuerza suficiente para mover al motor, que pone una fuerza de oposición de 5 N. Los resultados se encontraron por medio de experimentación, usando las siguientes ecuaciones:

Ecuaciones obtenidas de [40].

Datos:

$Material_{banda} = Caucho$…ecuación 166

$Material_{rueda} = Filamento\ PLA$…ecuación 169

$\mu s = Coeficiente\ de\ fricción\ estático$…ecuación 167

$\theta = Inclinación\ para\ encontrar\ \mu s$…ecuación 168

Cálculos estáticos:

$\sum F_x = 0$ …ecuación 172

$\sum F_y = 0$…ecuación 173

$\sum F_x = W \sin\theta - F_f = 0$ …ecuación 174

$F_f = \mu s(N)$…ecuación 175

$\sum F_y = N - W\cos\theta = 0$…ecuación 176

$N = W\cos\theta$…ecuación 177

$W\sin\theta = \mu s(W\cos\theta)$…ecuación 178

$\mu s = \frac{\sin\theta}{\cos\theta} = \tan\theta$…ecuación 179

El coeficiente de fricción estático μs es la tangente del ángulo de inclinación justo antes de que el cuerpo se empiece a mover y se encuentra específicamente para el rozamiento de los dos cuerpos que están en contacto durante el experimento.

Resultados estadísticos:

μs:

$\theta = 36°$ …ecuación 180

$\theta = 31°$ …ecuación 181

$\theta = 34°$ …ecuación 182

$\theta = 30°$ …ecuación 183

$\theta = 34°$ …ecuación 184

$\bar{\theta} = \frac{\sum_{i=1}^{N}\theta}{N} = 33°$…ecuación 185

$\mu s = \tan\theta = 0.6494$…ecuación 186

$\mu k = Coeficiente\ de\ fricción\ cinético$…ecuación 170

$F_f = Fuerza\ de\ fricción$…ecuación 171

Cálculos diámicos con inclinación θ mayor a la encontrada para μs:

$\sum F_x = m(a)$…ecuación 187

$\sum F_y = 0$…ecuación 188

$\sum F_x = m(g)\sin\theta - F_f = m(a)$ …ecuación 189

$F_f = \mu k(N)$…ecuación 190

$\sum F_y = N - m(g)\cos\theta = 0$…ecuación 191

$N = W\cos\theta = m(g)\cos\theta$…ecuación 192

$m(g)\sin\theta = \mu k(m(g)\cos\theta) + m(a)$…ecuación 193

$\mu k = \frac{\sin\theta}{\cos\theta} - \frac{a}{g\cos\theta} = \tan\theta - \frac{a}{g\cos\theta}$…ecuación 194

Para encontrar experimentalmente el coeficiente de fricción cinemático se toma en cuenta que la aceleración será constante por lo que se utiliza una de las fórmulas de aceleración constante:

$df = di + vi(t) + \frac{a(t^2)}{2} = 0 + 0 + \frac{a(t^2)}{2}$…ecuación 195

$a = \frac{2(df)}{t^2}$…ecuación 196

μk:

$\theta = 40° > 33°$…ecuación 197

$d = 0.123\ m$…ecuación 198

$t = 0.28\ s$ …ecuación 199

$t = 0.28\ s$ …ecuación 200

$t = 0.26\ s$ …ecuación 201

$t = 0.34\ s$ …ecuación 202

$t = 0.3\ s$ …ecuación 203

$\bar{t} = \frac{\sum_{i=1}^{N} t}{N} = 0.292$…ecuación 204

$a = \frac{2(df)}{t^2} = \frac{2(0.123)}{0.292^2} = 2.8851\ \left[\frac{m}{s^2}\right]$…ecuación 205

$$\mu k = \tan\theta - \frac{a}{g\cos\theta} = \tan 40° - \frac{2.8851}{9.81\cos 40°} =$$
0.4551…ecuación 206

El coeficiente de fricción estático y dinámico entre los materiales caucho y asfalto seco a una velocidad menor de 50 km/h, aplicados en la rueda trasera que es la que genera la tracción de la bicicleta son de [43]:

$\boldsymbol{\mu s = 0.75}$…ecuación 207
$\boldsymbol{\mu k = 0.55}$…ecuación 208

La posición del centro de masa de una bicicleta normal medido desde las ruedas frontal y trasera son las siguientes (figuras 72 y 73):

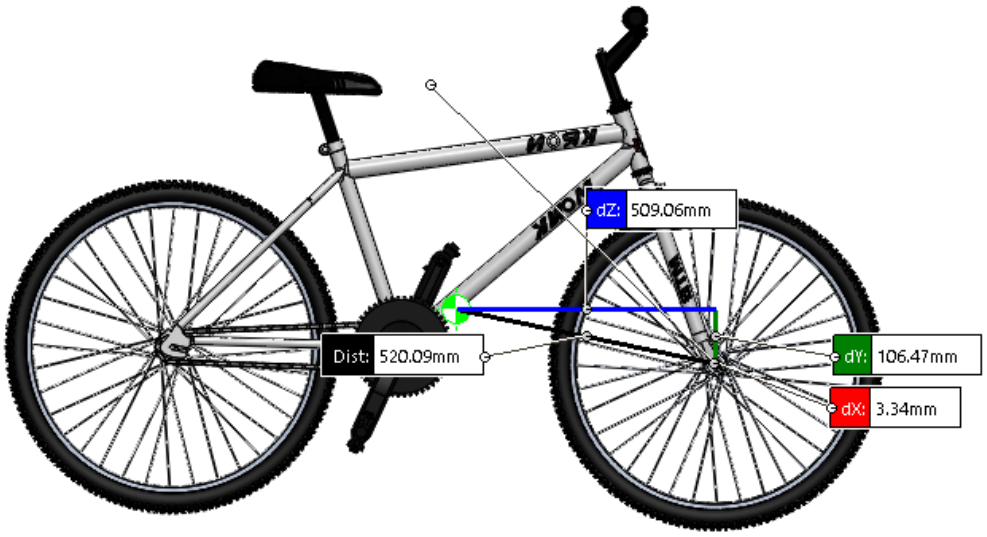


Figura 72. Centro de masa medido desde la rueda frontal.

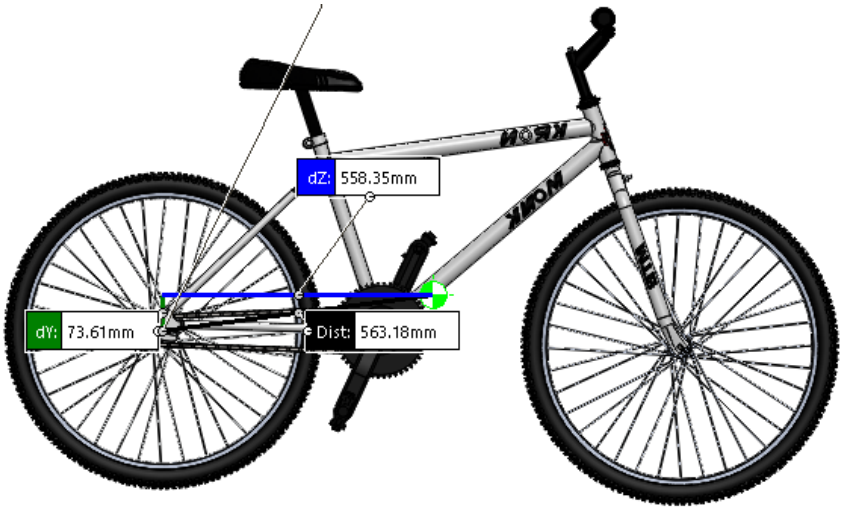


Figura 73. Centro de masa medido desde la rueda trasera.

La posición del centro de masa de una bicicleta Monk Kron R26 18V con la rueda frontal generadora de energía respecto a ambas ruedas es la siguiente (figuras 74 y 75):

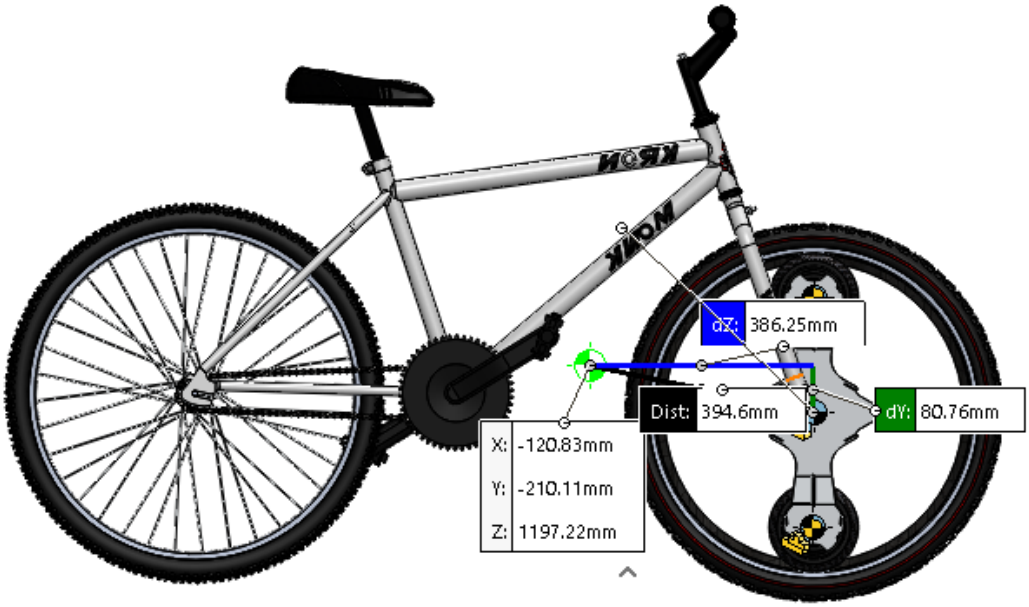


Figura 74. Centro de masa medido desde la rueda frontal.

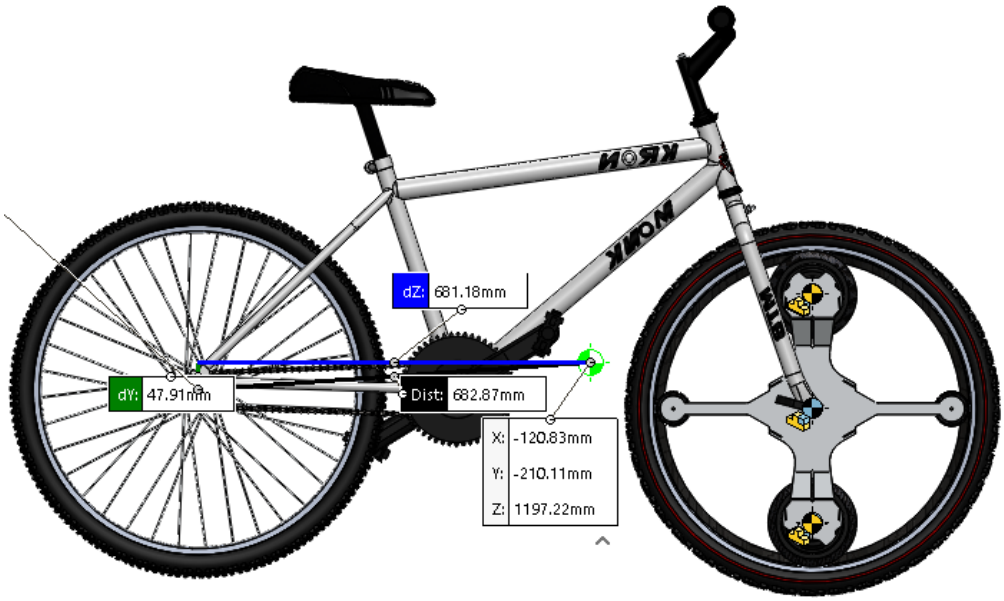


Figura 75. Centro de masa medido desde la rueda trasera.

El cálculo cinemático se realiza tomando en cuenta que la velocidad promedio de tránsito de bicicletas en la Ciudad de México es de 20 km/h = 5.5555 m/s:

Datos:

$N_t = Normal\ trasera$…ecuación 209

$N_d = Normal\ delantera$…ecuación 211

$W_u = Peso\ usuario = 100\ kg$…ecuación 213

$W_b = Peso\ bicicleta = 20\ kg$…ecuación 210

$F_f = Fuerza\ de\ fricción$…ecuación 212

$Pot_m = Potencia$…ecuación 214

Cálculos dinámicos – Bicicleta sin rueda frontal generadora de energía:

$\sum T_A = xcm(m(ay)) - ycm(m(ax))$…ecuación 215

$\sum F_y = 0$…ecuación 210

$a_x = \frac{Vf-V0}{t} = \frac{5.5555-0}{15} = 0.3703 \left[\frac{m}{s^2}\right]$…ecuación 216

$(0.49(-100 * 9.81)) + (0.5583(-20 * 9.81)) + 1.65N_d = 0 - (0.4366 * 20 * 0.3703) - (1.13 * 100 * 0.3703)$…ecuación 217

$N_d = 330.3946\ N$…ecuación 218

$\sum F_y = N_t + N_d - 120(9.81) = 0$…ecuación 219

$N_t = 846.8054\ N$…ecuación 220

$F_{fs} = \mu s(N_t) = 0.75(846.8054) = 635.104\ N$…ecuación 221

$F_{fk} = \mu k(N_t) = 0.55(846.8054) = 465.7429\ N$…ecuación 222

$Pot_m = F(vel) = F_{fk}(5.5555) = 465.7429(5.5555) = 2.5874\ kW$…ecuación 223

Cálculos dinámicos – Bicicleta con la rueda frontal generadora de energía:

$\sum T_A = xcm(m(ay)) - ycm(m(ax))$…ecuación 224

$\sum F_y = 0$…ecuación 225

$(0.49(-100 * 9.81)) + (0.6811(-20 * 9.81)) + 1.65N_d - 1.2925(2) = 0 - (0.4109 * 20 * 0.3703) - (1.13 * 100 * 0.3703)$…ecuación 226

$N_d = 346.6786\ N$…ecuación 228

$\sum F_y = N_t + N_d - 120(9.81) = 0$…ecuación 229

$N_t = 830.5214\ N$…ecuación 230

$F_{fs} = \mu s(N_t) = 0.75(830.5214) = 622.891\ N$…ecuación 231

$F_{fk} = \mu k(N_t) = 0.55(830.5214) = 456.7867\ N$…ecuación 232

$Pot_m = F(vel) = F_{fk}(5.5555) = 456.7867(5.5555) = 2.5376\ kW$…ecuación 233

Esta potencia mecánica de entrada será dividida entre la potencia eléctrica de salida para obtener la eficiencia del sistema generador de energía.

A continuación, se realizará el análisis de energías para determinar si el buje soporta cierto impacto directo y a que altura considerando su deflexión máxima y que su material es aluminio 6061-T6:

Datos:

$N_d = 346.6786\ N$…ecuación 234

$mi = 17.6696\ kg = masa\ impacto$…ecuación 235

$a = ancho = 20.2\ mm$…ecuación 236

$h = altura\ impacto$…ecuación 237

$de = diámetro\ exterior = 12\ mm$…ecuación 238

$di = diámetro\ interior = 5\ mm$…ecuación 239

$L = Largo\ viga = 43.9\ mm$…ecuación 240

$E = Módulo\ de\ elasticidad = 70X10^9$…ecuación 241

$\sigma pc = Esfuerzo\ de\ cedencia = 240\ MPa$…ecuación 242

$g = 9.81\ \frac{m}{s^2}$…ecuación 243

Cálculo del momento de inercia Izz y determinación de la ecuación de la elástica E(Izz)y'' = M(x):

$Izz = \frac{\pi}{64}(de^4 - di^4) = \frac{\pi}{64}(0.012^4 - 0.005^4) = 9.8719X10^{-10}\ mm^4$…ecuación 244

Balanceo de energías de deformación y potencial siguiendo la ley de conservación de la energía:

$U = \frac{W(ymax)}{2}$…ecuación 255

$W = mg(h + ymax)$…ecuación 256

$E(Izz)y'' = M(x)$…ecuación 245

$M(x) = w\langle x\rangle - wL\langle x\rangle^0$…ecuación 246

$E(Izz)y'' = w\langle x\rangle - wL\langle x\rangle^0$…ecuación 247

Procedemos a ejecutar el método de la doble integración

$E(Izz)y' = \frac{w}{2}\langle x\rangle^2 - wL\langle x\rangle^1 + C1$…ecuación 248

$E(Izz)y = \frac{w}{6}\langle x\rangle^3 - \frac{wL}{2}\langle x\rangle^2 + C1\langle x\rangle + C2$…ecuación 249

Posteriormente evaluamos en condiciones de frontera: x=0; y=0, x=0; θ=0

$0 = 0 - 0 + C1$…ecuación 250

$0 = 0 - 0 + 0 + C2$…ecuación 251

Obteniendo finalmente la ecuación de deflexión máxima, donde x=L, donde el signo negativo solo indica que la deflexión tiene dirección hacia abajo

$E(Izz)ymax = -\frac{wL^3}{3}$…ecuación 252

$ymax = \frac{wL^3}{3E(Izz)}$…ecuación 253

$w = \frac{ymax(3)E(Izz)}{L^3}$…ecuación 254

$U = \frac{3E(Izz)ymax^2}{2L^3} = mg(h + ymax)$…ecuación 257

$h = \frac{3E(Izz)ymax^2}{2L^3mg} - ymax$…ecuación 258

Ahora para obtener la deflexión máxima se considera la fórmula de esfuerzo

$\sigma = \frac{Mmax(c)}{Izz} = \frac{w(L)c}{Izz} = \frac{ymax(3)E(Izz)c}{Izz(L^2)}$…ecuación 259

$c = \frac{de}{2}$…ecuación 260

$ymax = \frac{\sigma L^2}{3E(c)} = 3.6708X10^{-4}$…ecuación 261

$h = \frac{3E(Izz)ymax^2}{2L^3mg} - ymax = 5.8532X10^{-4}$…ecuación 262

$w = \frac{ymax(3)E(Izz)}{L^3} = 899.471N$…ecuación 263

Para comprobar se realiza la operación en ambos lados de la ecuación de conservación de energía y podemos observar que dan como resultado lo mismo.

$\frac{W(ymax)}{2} = mg(h + ymax) = 0.165$…ecuación 264

Esto demuestra teóricamente que el buje resiste un impacto directo ya que el resultado es positivo al recibir toda la fuerza normal delantera y caer cierta altura h, el resultado de la deflexión máxima es respaldado por la simulación como se puede ver en la figura 76.

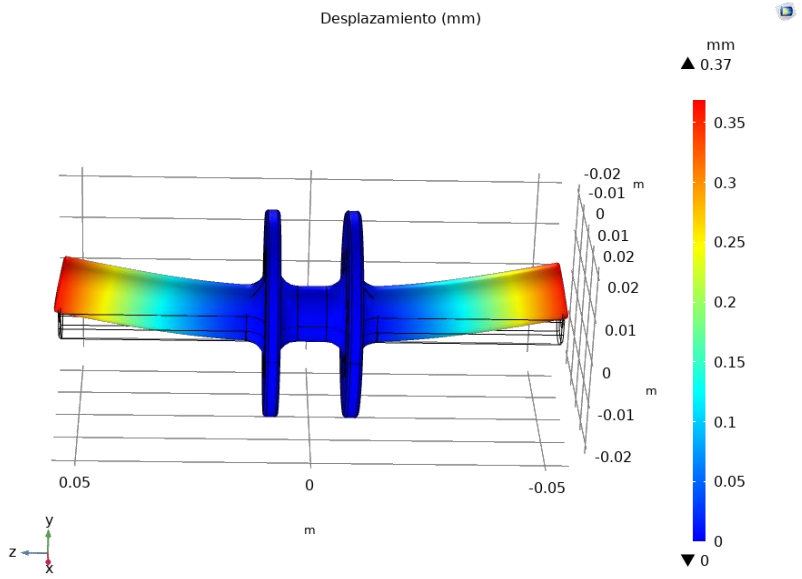


Figura 76. Deflexión máxima de buje – Ancho de 20.2 mm – Esfuerzo del punto de cedencia: 240 MPa.

Pero hay que tomar en cuenta que en la rueda frontal se tiene un amortiguamiento dado por la llanta, por lo que el impacto no será directo, además dentro del buje está el eje de cierre rápido que es de acero A36, por lo que ayudará a soportar un impacto. El umbral para determinar en la simulación si el buje soporta un impacto es aplicar el doble de carga estática a la viga y se puede observar en la siguiente figura que si lo soporta con un factor de seguridad mayor a 1 (figura 77).

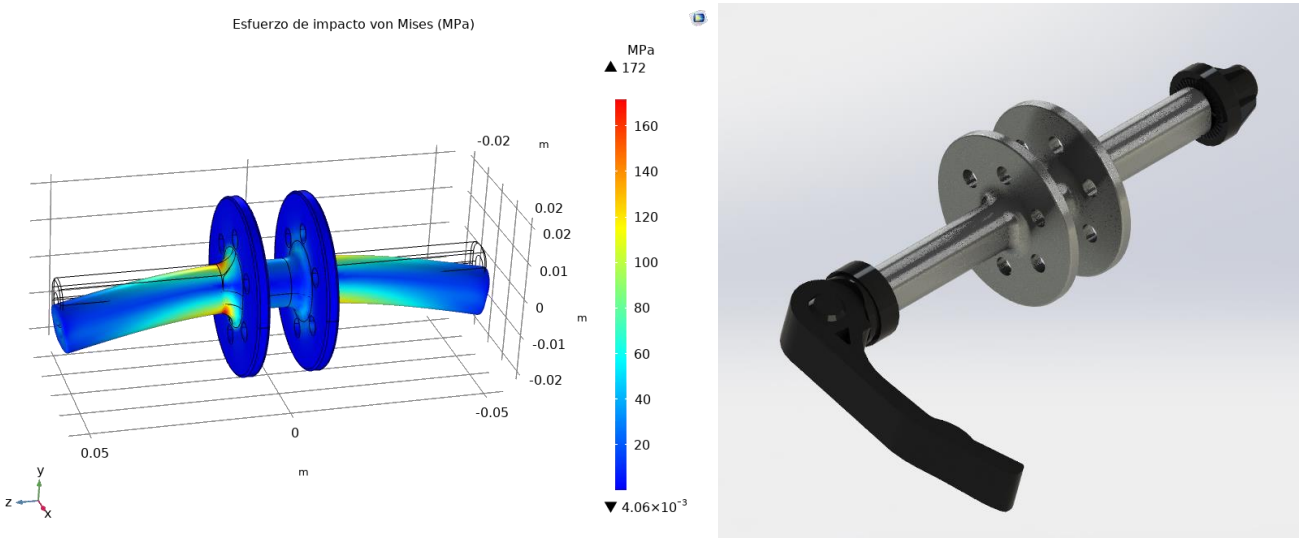


Figura 77. Esfuerzo con impacto – Ancho de 20.2 mm – Esfuerzo máximo: 85.7 MPa.

$$\sigma max = 172\ MPa \text{…ecuación 265}$$

$$F.S. = \frac{\sigma_{pc}}{\sigma_t} = \frac{240}{172} = 1.3953 \text{…ecuación 266}$$

d.2.2. Potencia eléctrica de salida del generador MXUS-XF07

Se llevó a cabo la caracterización del generador MXUS-XF07, el cual se uso las salidas del motor conectadas a un puente de diodos rectificador trifasico, el cual se midio su salida, como se muestra en la figura 78:

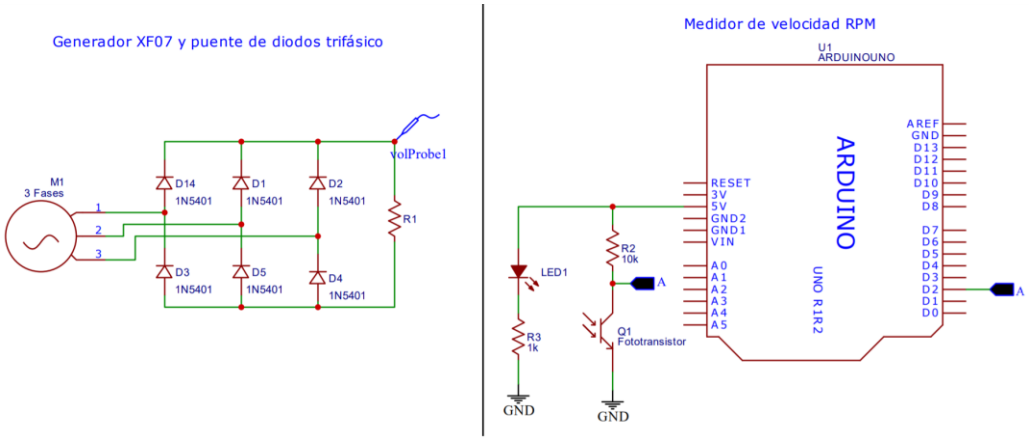


Figura 78. Circuito de medición para obtener de la potencia eléctrica de salida.

Primero se realizo la caracterización del motor sin ninguna carga, los resultados son los mostrados en la tabla 44:

Tabla 44. Caracterización del generador MXUS-XF07.

RPM	Tensión de Salida [V]
66	13.22
150	25.43
168	29.43
240	40.09
294	50.02
366	62.56

Los valores de la tensión y rpm son valores promedio, los cuales fueron determinados al girar el generador a una velocidad constante. En la figura 79 se muestran los datos estadísticos recabados y la ecuacion característica del generador sin carga.

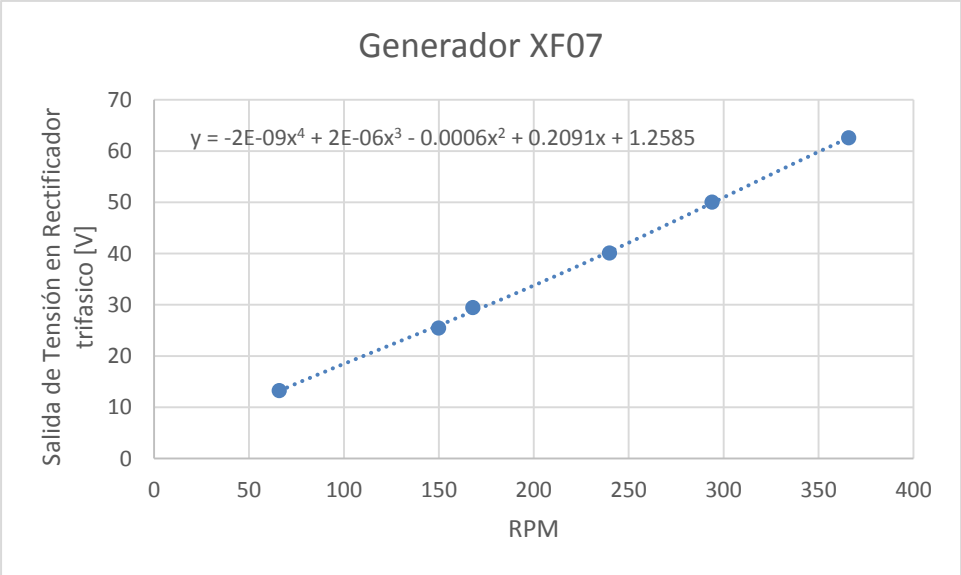


Figura 79. Caracterización del motor MXUS-XF07 en modo generador sin carga.

A continuación, en la tabla 45 se muestran los valores obtenidos al girar el motor con una carga de 27.3 ohms, que es el valor de la carga de la batería de bicicleta:

Tabla 45. Tesiones de salida del rectificador de un generador con una carga a la salida de 27.3ohms.

RPM	Tensión de Salida [V]
42	3.33
48	5.8
78	8.09
84	9.99
90	10.37
96	13.97

La grafica de la figura 80 muestra los puntos resultantes:

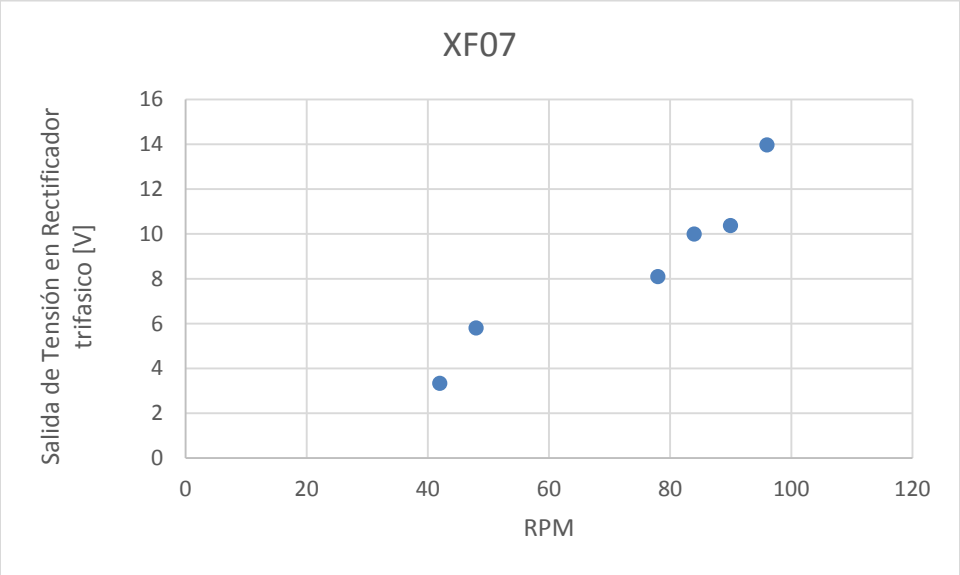


Figura 80. Caracterización del motor MXUS-XF07 en modo generador con carga de la batería de bicicleta eléctrica.

d.3. Sistema de captación y acondicionamiento de energía eléctrica

De acuerdo con el circuito diseñado anteriormente, se tomó la decisión de optar por un modelo comercial de convertidor CC elevador-reductor debido a que hacer el diseño de una fuente conmutada desde cero es mucho más ineficiente que utilizar una del mercado, ya que algunos elementos se tienen que mandar a hacer especialmente para el circuito, como lo es el devanado.

Primero se seleccionó un convertidor reductor/elevador, para que pueda controlarse la tensión entregada ya sea para para alimentar una batería portátil con entrada de 5V o una batería de bicicleta eléctrica con entrada de 54.6V, ambas con corriente nominal de 2A. En tal caso se revisaron los diferentes

convertidores reductores/elevadores del mercado y no se en encontró uno adecuado para proporcionar una tensión de salida de 54.6 V o siquiera aproximarse.

Por lo cual, la solución es usar un convertidor elevador (del inglés "*buck"*) para poder llegar a alimentar a esa tensión de 54.6V y después utilizar un convertidor reductor (del inglés "*boost"*), para poder alimentar la batería portátil 5V.

Para seleccionar un convertidor comercial necesario, se tomó en cuenta los siguientes requerimientos enlistados en la tabla 46.

Tabla 46. Orden de los objetivos más importantes.

Objetivos	Rango de operación de Tensión	Corriente [A]	Potencia	Eficiencia	Dimensiones	Precio	Suma
Rango de operación de Tensión		1	0.5	0.5	1	1	**4**
Corriente [A]	0		0.5	0.5	1	1	**3**
Potencia	0.5	0.5		0.5	1	1	**3.5**
Eficiencia	0	0.5	0.5		1	1	**3**
Dimensiones	0	0	0	0		0.5	**0.5**
Precio	0	0	0	0	0.5		**0.5**

En la tabla 47, se muestran los convertidores más aptos en el mercado con sus datos técnicos, además se realiza su calificación y suma de valores, multiplicados por su peso de asignación para ver cuál es el más optimo:

Tabla 47. Selección del convertidor elevador.

Objetivo	Pesos de asignación	Convertidor 1 Valores Técnicos	Convertidor 1 Calificación	Valor	Convertidor 2 Técnicos	Convertidor 2 Calificación	Valor
Rango de operación de Tensión [V]	0.25	Ventrada = [8 - 60] Vsalida = [12 - 80]	8	2	Ventrada = [8.5 - 60] Vsalida = [10 - 60]	6	1.5
Potencia [W]	0.25	1200	8	2	400	6	1.5
Corriente [A]	0.15	20 máximo	8	1.2	8	8	1.2
Eficiencia	0.15	95%	7	1.05	96%	8	1.2

Dimensiones [mm]	0.1	130x52x53mm	7	0.7	67mmx48mmx 28mm	8	0.8
Precio	0.1	313	8	0.8	555	5	0.5
Peso ponderado				**7.75**			**6.7**

En la figura 81 se muestra el convertidor elevador (del inglés "*boost"*) seleccionado, después de realizar pruebas se determinó que se activa con una tensión mínima de 6.8V.

Figura 81. Convertidor CC-CC Boost.

Para alimentar la batería portátil se propuso usar un convertidor reductor (del inglés "*buck"*), este convertidor viene con un regulador LM2596 (figura 82), especializado para convertidores reductores. Este reductor recibe como entrada de 3.2 a 40 Volts, dando como salida desde 1.25 hasta 37 Volts. Su eficiencia es de 92% y para activarse necesita una tensión mínima de 6V.

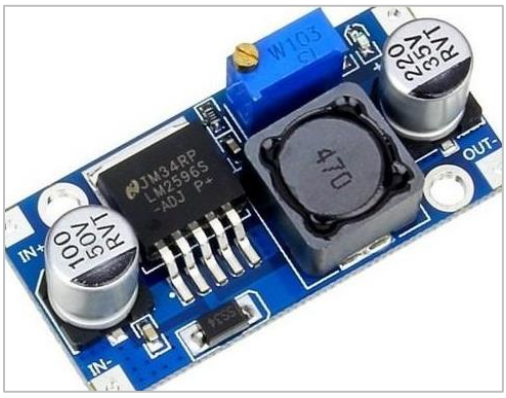


Figura 82. Convertidor CC-CC Buck.

Se plantea controlar la alimentación ya sea para alimentar la batería portable o la batería de bicicleta con ayuda de un relevador. La opción de alimentar la batería portátil será la opción que se encontrará por defecto con el relevador, al activar el relevador cambiará a alimentar la batería de bicicleta.

Se utilizaron dos puentes de diodos porque existen dos generadores, y se agregó un capacitor para evitar la caída de tensión por la generación discontinua. Posteriormente pasa la señal al convertidor comercial

elevador para entregar una tensión de 54.6 Volts o dependiendo del control del microcontrolador, podría cambiar a otra tensión.

El convertidor reductor que controla o reduce la tensión para poder alimentar la batería portátil. El usuario tiene la opción de alimentar una batería de bicicleta o una batería portátil, esto es posible por medio del dispositivo móvil del usuario, el cual por medio de una conexión inalámbrica Bluetooth le es posible enviar comandos al microcontrolador ESP32. El microcontrolador recibe la orden y abre o cierra compuertas con los Mosfet para obtener una tensión o la otra.

Los Mosfet trabajan en corte y saturación además de eso se les agrego una amplificación contra fásica (del inglés "*Push-Pull*"). Esto hace que los Mosfet eviten estar tanto tiempo en la zona óhmica y así prevenir que se caliente el transistor, dando toda la energía a la carga, que ente caso son las baterías.

Entrada de generadores

En esta parte del circuito, se conectarán los dos generadores en los terminales del TBLOCK como se ve en la figura 83, después pasarán a un puente de diodos para rectificar la señal trifásica, para finalmente llegar a un capacitor y así mantener la señal lo más estable posible.

El máximo valor de capacitancia permitido es de 120uF porque los valores pico-pico de corriente son de 812mA, si se aumenta a otro valor de capacitancia comercial próximo, las corrientes pico-pico alcanzarían más de 2A, lo cual puede dañar a los diodos rectificadores.

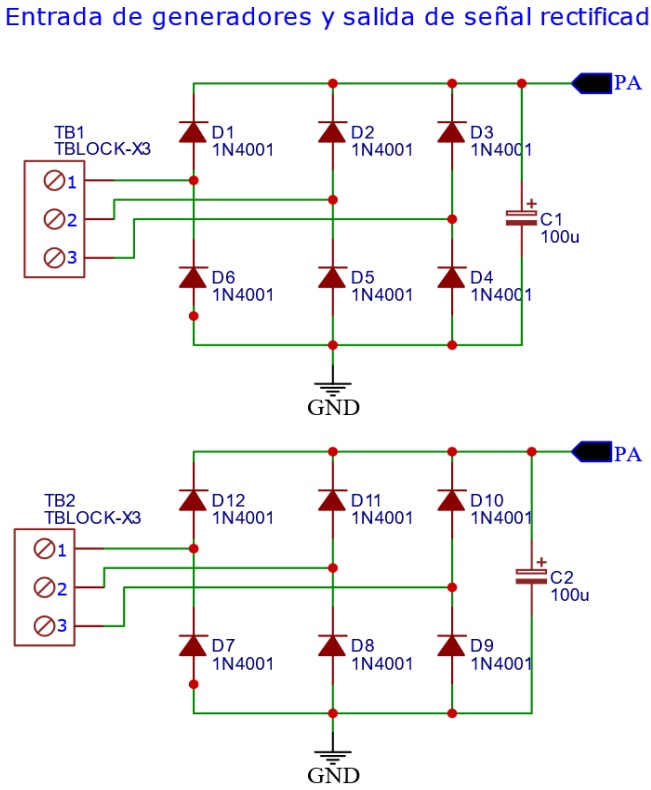


Figura 83. Entrada de generadores y salida de señal rectificada.

Control de señal rectificada

Esta parte del circuito (figura 84) llega la señal rectificada a un relevador para seleccionar la terminal de salida, las salidas llevan a dos convertidores, uno convertidor elevador y convertidor reductor.

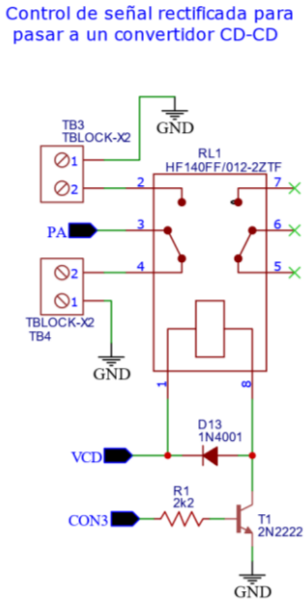


Figura 84. Control de señal rectificada para pasar a un convertidor CD-CD.

A continuación se muestran los cálculos para el diseño de esta parte del circuito.

Se utilizó un relevador de 9V con una resistencia de 160 Ω. Calculando la corriente obtenemos:

$I_c = \frac{9V}{160\Omega} = 56.25mA$…ecuación 267

Por lo tanto, se utilizó un transistor que soportara la corriente de 56.25 mA, por lo que se seleccionó un transistor 2N2222A. El transistor tiene una hfe de 100 a150. Por lo tanto, se propuso una hfe de 100.

Con la ecuación de la corriente de base y colector se obtiene:

$I_c = \beta I_B \rightarrow I_B = \frac{56.25mA}{100} = 0.56mA$…ecuación 268

Se incrementó el valor de la corriente de base porque ese valor del resultado es un valor critico mínimo para suministrar la corriente de 56.25mA.

Por lo tanto, la corriente es $I_B = 1.12mA$

…ecuación 269

Con la siguiente ecuación se puede determinar la resistencia R_B:

$R_B = \frac{3.3-0.7}{1.12mA} = 2321.42\Omega$…ecuación 270

La resistencia comercia aproximada es:

$R_B = 2.2k\Omega$…ecuación 271

Se añadió un diodo 1N4001 para evitar que la bobina del relevador se descargue y pueda dañar el transistor.

Salida para la alimentación de la batería

Para esta parte del circuito, su entrada en TB6 proviene de un convertidor reductor al cual se le entrego la energía que provenía del puente de rectificación y filtrado. Este parte del circuito tiene un divisor de tensión para dar una lectura de la tensión al microcontrolador (figura 85).

El microcontrolador ESP32 solo permite lecturas analógicas de hasta 3.3V por lo cual se utilizó un divisor de tensión, el cual se muestra en la figura 74, Para R1 el resultado fue 33,700Ω y para R2 es de 3,300Ω. Pero las resistencias obtenidas no son comerciales, por lo que se usó las resistencias que existen en el mercado para hacer una aproximación.

$$R_1 = 150\Omega + 560\Omega + 33\text{k}\Omega = 33710\Omega \ldots \text{ecuación 272}$$

$$R_2 = 3.3k\Omega \ldots \text{ecuación 273}$$

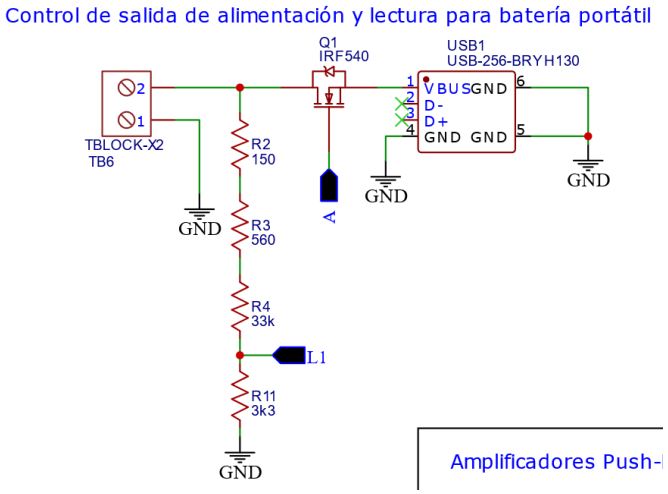


Figura 85. Salida para alimentar la batería portátil de 5V.

Como se puede observar en la figura anterior, se toma la lectura en la terminal L1, estas lecturas pueden llegar a tomar una relación de 3.3V, que en la salida del convertidor reductor llega a un límite máximo de 37V.

Salida para la alimentación de la batería de bicicleta

En esta parte del circuito (figura 86) entra la señal de un convertidor elevador, la cual proporciona una tensión de 54V, para poder verificar la tensión se utilizó un divisor de tensión para que el microcontrolador pueda leer una relación de la señal. La lectura máxima que puede realizar por parte del divisor de tensión es de 83V, que es lo máximo que se puede generar por medio del convertidor elevador. Las resistencias calculadas para el divisor de tensión son las siguientes, $R_1 = 79700\Omega$ y $R_2 = 3.3k\Omega$. Las resistencias comerciales son:

$$R_1 = 56k\Omega + 22\text{k}\Omega + 1.8\text{k}\Omega = 79800\Omega \ldots \text{ecuación 274}$$

$$R_2 = 3.3k\Omega \ldots \text{ecuación 275}$$

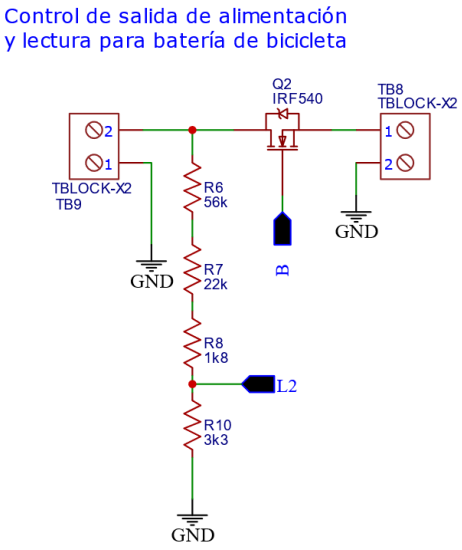


Figura 86. Salida para alimentar la batería de bicicleta.

En las dos figuras anteriores se muestra que se está utilizando el transistor Mosfet como interruptor, el cual se añadió por seguridad para que el microcontrolador realice la lectura de la tensión y solamente si es correcta la tensión medida, proceda a abrir el interruptor.

Activacion de Mosfet como interruptor

Se utilizaron dos mosfet como interruptores en caso de que la tensio de salida de los convertidores cuando esten fuera de rango. Uno es para la parte de la salida para alimentar la bateria portatil de 5V. En la siguiente figura 87, se muestra el circuito para controlar el primer mosfet IRF540.

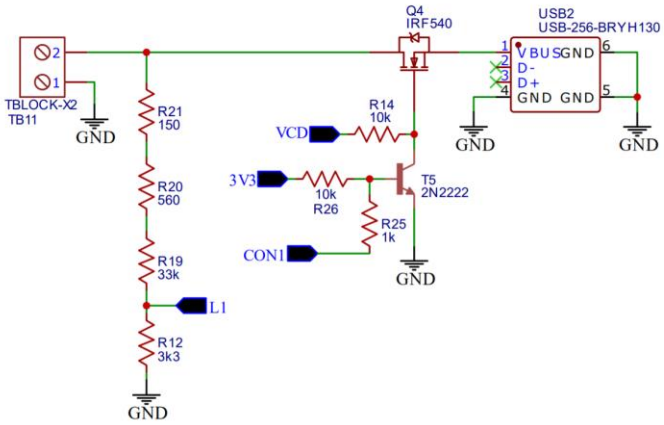


Figura 87. Control del mosfet para utlizarlo como interruptor.

Como ya es de saber el mosfet se controla aplicando tensión en su entrada llamada puerta (del ingles *gate*). En la hoja de dato se puede controlar a partir de una tensión de 2 a 4 volts de tensión, la variable llamada $V_{GS(TH)}$, pero a estos valores de tensión puede ocasionar problemas por la caida de tensión entre

los entradas de drenador y fuente, otro punto importante es el aumento de temperatura en el componente y la zona ohmica. Para evitar esos factores lo recomendable es aumenta la tensión que se le suministra en el porton.

En el anexo numero 7, se muestra una grafica caracteristica del IRF540N de saturación, podemos observar que entre mas se aumente la tensión en el porton, se evitar pasar la zona ohminca del mosfet, se evita la caida de tension, el aumento de temperatura. Por lo cual se realizo la siguiente configuración que se muestra en la figura 88.

Se utilizo un transistor 2N2222, el cual se satura o se encuentra en un circuito abierto dependiento si la señal CON3 de la ESP32, si esta en parte logica alta, el mosfet se encuentra como un circuito abierto, y si la señal de la ESP32 esta en logica baja, el mosfet se encuentra como un circuito cerrado.

Para el control del mosfet que controla la tension de salida para alimentar la bicicleta se utilizo el siguiente diagrama. En este cierto solo cambia con la resistencia de 100K que se pone de la salida de tension de 54V a el porton del mosfet. Se realiza la misma logica con la señal que entrega la ESP32 en el puerto CON2.

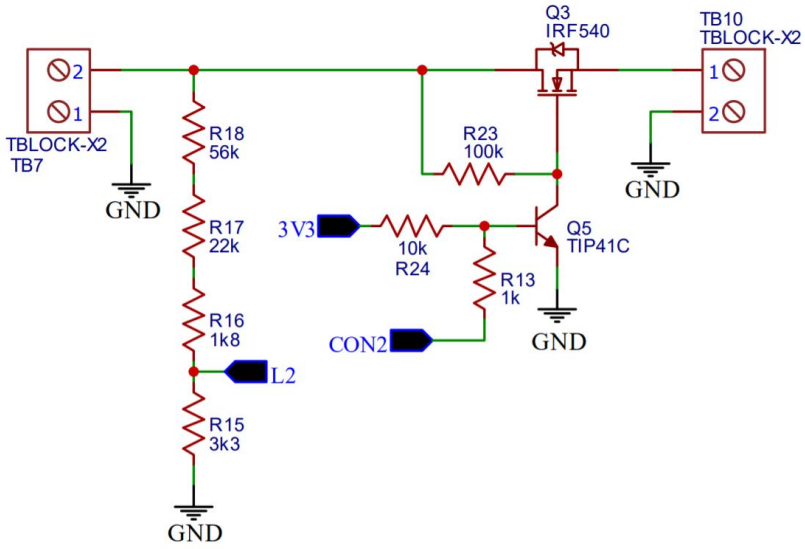


Figura 88. Control del mosfet como interruptor de salida para alimentar la bateria de la bicicleta.

Zona de control y alimentación del circuito eléctrico

Para esta parte del circuito se utilizaron los pines D34 y D33 (CON1 y CON2) del microcontrolador para poder controlar la apertura y cierre de los Mosfet para brindarle alimentación a las baterías.

Para leer las lecturas de tensión a la salida de los convertidores se utilizó los puertos D35 y D33 (L1 y L2).

Con el pin D2 (CON3) se controla el relevador. El puerto VIN se alimenta a 9V el microcontrolador ya que tiene dentro de su placa un regulador ASM1117, con el cual se le puede suministrar hasta 12V de tensión.

En la parte izquierda de la figura 89, se conecta una batería de 9V para alimentar el microcontrolador y los elementos del circuito.

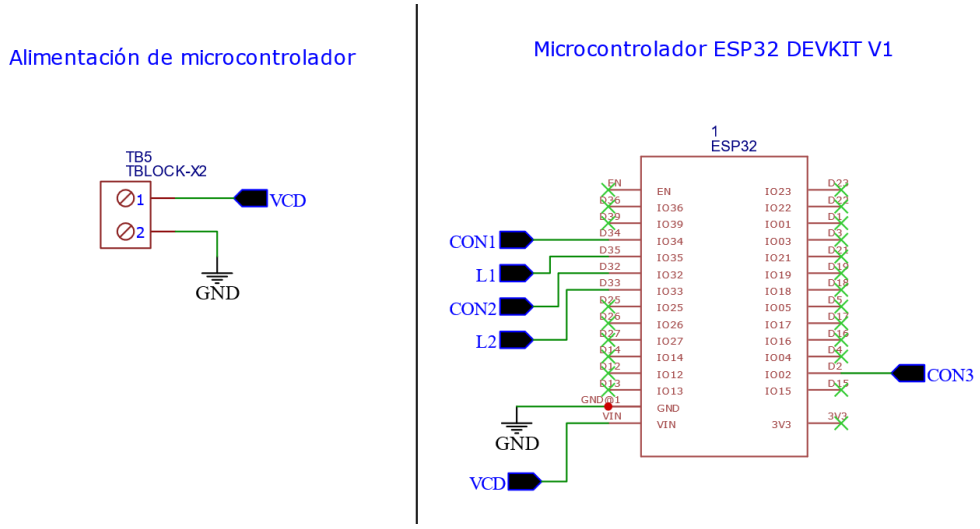


Figura 89. Alimentación de ESP32 y salidas - entradas del microcontrolador.

Medidor de carga de batería de litio

Se realizará la medición de la batería de litio de la batería portátil que se utilizará en este proyecto. La batería portátil contiene dentro una batería de polímero de litio modelo 606090. Por lo cual, la hoja de datos de la batería que se encuentra en el **Apéndice D** contiene su curva característica de carga y descarga con respecto a los niveles de tensión.

Por lo tanto, se procedio a realizar el ajuste de una ecuación que describa la carga con respecto a la tensión mostrada a continuación:

$$y = 0.01119 * \mathrm{X}^{6.714} - 14.47 \ldots \text{ecuación 276}$$

Los valores de tensiones que maneja la batería van de los 3 hasta los 4.2V. Por lo tanto, se procedio a usar un divisor de tensión para que el microcontrolador pueda realizar su lectura.

Se propuso un divisor de tensión con tensión máxima de entrada de 5V, para que a la salida dé una tensión de 3.3V. Con el fin de evitar un sobrevoltaje en el microcontrolador ESP32. El resultado de las resistencias R1 y R2 al realizar el divisor de tensión son:

$$R_1 = 22k\Omega \ldots \text{ecuación 277}$$

$$R_2 = 33k\Omega \ldots \text{ecuación 278}$$

En la figura 90 se muestra el orden de las resisitencias comerciales con las cuales se realizó el divisor de tensión, la entrada CN1 para medir la tensión de la batería de litio y la tensión reajustada para medirse en L3 por parte del microcontrolador. Finalmente, el microcontorlador realiza el cálculo con la ecuación 133 y manda el dato de la carga en forma de porcentaje al dispositivo móvil del usuario.

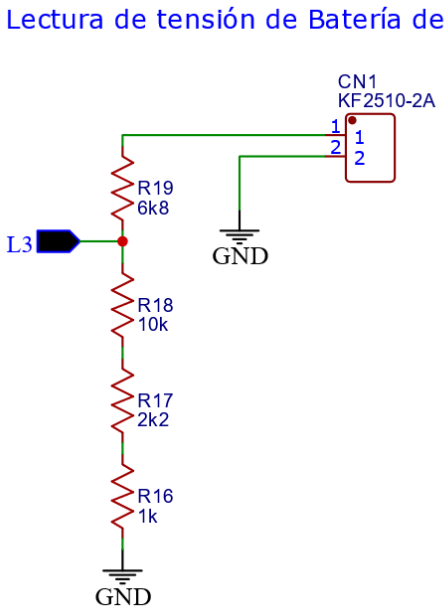


Figura 90. Lectura de tensión para determinar el porcentaje de carga de la batería de litio.

d.4. Aplicación móvil en sistema operativo Android

Se desarrolla el código que monitoree la velocidad, tiempo, distancia y ruta del usuario en Android, dentro del software Android Studio.

Las páginas creadas en el entorno de desarrollo llamado Android Studio son responsivas, esto implica que la interfaz de la aplicación se puede adaptar a los distintos grosores de la pantalla donde se esté viendo la aplicación, ya sea si se está viendo en un celular con pantalla grande, pantalla chica, televisiones, tabletas, etc.

En el **Anexo 2** se muestra el código realizado y documentado en el editor de texto Android Studio para el desarrollo de la aplicación móvil.

A continuación, se describen las librerías y APIs utilizadas dentro del proyecto creado para la aplicación móvil.

Material Design

Material Design es una librería que asiste en el diseño visual, interactivo y de movimiento en las aplicaciones desarrolladas con Android Studio.

Componentes de Arquitectura ViewModel

Se utiliza la clase ViewModel a fin de almacenar y administrar datos relacionados con la interfaz de usuario de manera optimizada para los ciclos de vida de los elementos en una clase de Kotlin. La clase ViewModel permite que se conserven los datos luego de cambios de configuración, como las rotaciones de pantallas.

Almacenamiento ROOM

El almacenamiento por medio de las bases de datos Room proporcionan una capa de abstracción sobre SQLite, que permite acceder a la base de datos almacenada dentro del mismo dispositivo móvil donde esté corriendo la aplicación, en vez de almacenarse en un servidor remoto.

Corrutina

Una corrutina es un patrón de diseño de simultaneidad que se puede utilizar en Android Studio para simplificar el código que se ejecuta de forma asíncrona. Las corrutinas se utilizan para correr algunas instrucciones en paralelo, en vez de hacerlo en forma secuencial, ayudando a administrar tareas de larga duración que, de lo contrario, podrían bloquear el subproceso principal y hacer que la aplicación dejara de responder.

Navigation

Navigation es una librería utilizada para navegar entre las "páginas" dentro de una misma aplicación Android, ya sea que los destinos se implementen como fragmentos, que es una parte dinámica de la aplicación, como actividades o algún otro tipo de componente.

Básicamente por medio de la librería Navigation se puede crear un menú de navegación que navege por los distintos fragmentos de una aplicación Android, en vez de realizar la navegación por medio de las actividades (que es como se le llama al código que define cada página de la aplicación).

El menú de la aplicación navega por medio un gráfico de navegación, que es un archivo gráfico que contiene todos destinos y acciones a través de los cuales puede navegar la aplicación.

Glide

Glide es una popular librería Android de código abierto que sirve para cargar imágenes, videos y GIFs animados de forma optimizada, sin que se comprometa la velocidad de la aplicación.

Datos de ubicación - Google Maps Location Services:

Una de las funciones exclusivas de las aplicaciones para dispositivos móviles es el conocimiento de la ubicación. Por eso, si se agrega la función de conocimiento de la ubicación a la aplicación Android, se podrá hacer el seguimiento de la ruta que recorra el usuario.

Entre los datos de ubicación disponibles para un dispositivo Android, se incluyen la ubicación actual del dispositivo (identificada a través de una combinación de tecnologías), la dirección y el método de movimiento, y si el dispositivo cruzó un límite geográfico predefinido, lo que también se conoce como perímetro virtual.

Dagger

La inserción manual de dependencias o localizadores de servicios como la API de Google Maps en una aplicación Android puede ser problemática, por lo cual se puede limitar la complejidad de la importación de librerías mediante el uso de Dagger para administrar dependencias. Dagger genera automáticamente un código que imita el código que se hubiera escrito a mano para utilizar ciertas librerías.

Timber

Es una biblioteca de código abierto que proporciona funciones de registro (del inglés *"Login"*).

MPAndroidChart

MPAndroidChart es una biblioteca de gráficos de código abierto que permite no solo dibujar varios gráficos estadísticos en un dispositivo Android, sino también arrastrar y enfocar en ellos, con varios tipos de gráficas de uso común como: gráfico de líneas, gráfico circular, histograma y gráfico de dispersión.

Ciclos de vida optimizados (lifecycle)

Los ciclos de vida se refieren al momento en donde ciertos datos almacenados en algunas variables van a ser reiniciados, como cuando se cierra la aplicación o cuando simplemente se cambia de pantalla. Los componentes optimizados para ciclos de vida realizan acciones como respuesta a un cambio en el estado del ciclo de vida de otro componente, como actividades o fragmentos. Estos componentes ayudan a crear un código mejor organizado y más liviano, que resulta más fácil de mantener y permite que la aplicación sea más rápida.

Partes del proyecto Android y sus funciones:

Carpeta /app/src/main/AndroidManifest.xml. Es un archivo XML que contiene todos los permisos necesarios que se debe pedir al usuario para acceder a los distintos sensores del dispositivo móvil para obtener acceso a la cámara, GPS, etc.

Carpeta /app/src/main/java. Esta carpeta contiene todo el código fuente de la aplicación, clases auxiliares, etc. La carpeta contiene el archivo principal del proyecto Android llamado Actividad principal (del inglés *"Main Activity"*), es un archivo de extensión y lenguaje Kotlin que funciona como la clase principal y maneja todas las funcionalidades del proyecto, conectándose con las demás clases para integrarlas entre sí.

Carpeta /app/src/main/res/. Contiene todos los recursos necesarios para el proyecto como: imágenes, zonas de la interfaz gráfica (del inglés *"layouts"*), cadenas de texto, etc.

Fichero /app/build.gradle. Contiene toda la información necesaria para la compilación del proyecto, por ejemplo, la versión del SDK de Android, la mínima versión de Android que soporta la aplicación, referencias a las librerías externas utilizadas, etc. En este archivo del proyecto se importan las librerías que se quiera usar dentro de la aplicación Android, específicamente esto se realiza en la parte del código que dice dependencias (del inglés *"dependencies"*).

app/java/res/layout/activity_main.xml. En este archivo del proyecto con extensión XML es donde se editan los aspectos visuales del proyecto. Aunque XML es un lenguaje de marcado que principalmente sirve para la trasmisión de datos, en Android se utiliza para colocar elementos gráficos como botones, conjunto de elementos pertenecientes a la interfaz gráfica (del inglés *"layouts"*), áreas de texto, etc.

app/java/res/values/strings.xml. En este archivo del proyecto se indica el texto que aparece por defecto en la parte superior de la pantalla de la aplicación Android, además en este archivo se puede indicar que deje de aparecer dicho texto.

app /java/res/values/colors.xml. En este archivo del proyecto se indican los colores que conforman las partes de la aplicación Android.

e. Integración de áreas funcionales

Para realizar la integración de áreas funcionales se hace uso del diagrama IDEF0 para desarrollar la arquitectura física del sistema.

e.1. IDEF0

La herramienta IDEF0 (del inglés *Integration Definition for Function Modelling*) permite representar gráficamente el comportamiento del sistema y la interacción de los elementos funcionales que lo integran. A continuación, se muestra la representación gráfica del IDEF0 (figura 91), para determinar la arquitectura física del sistema.

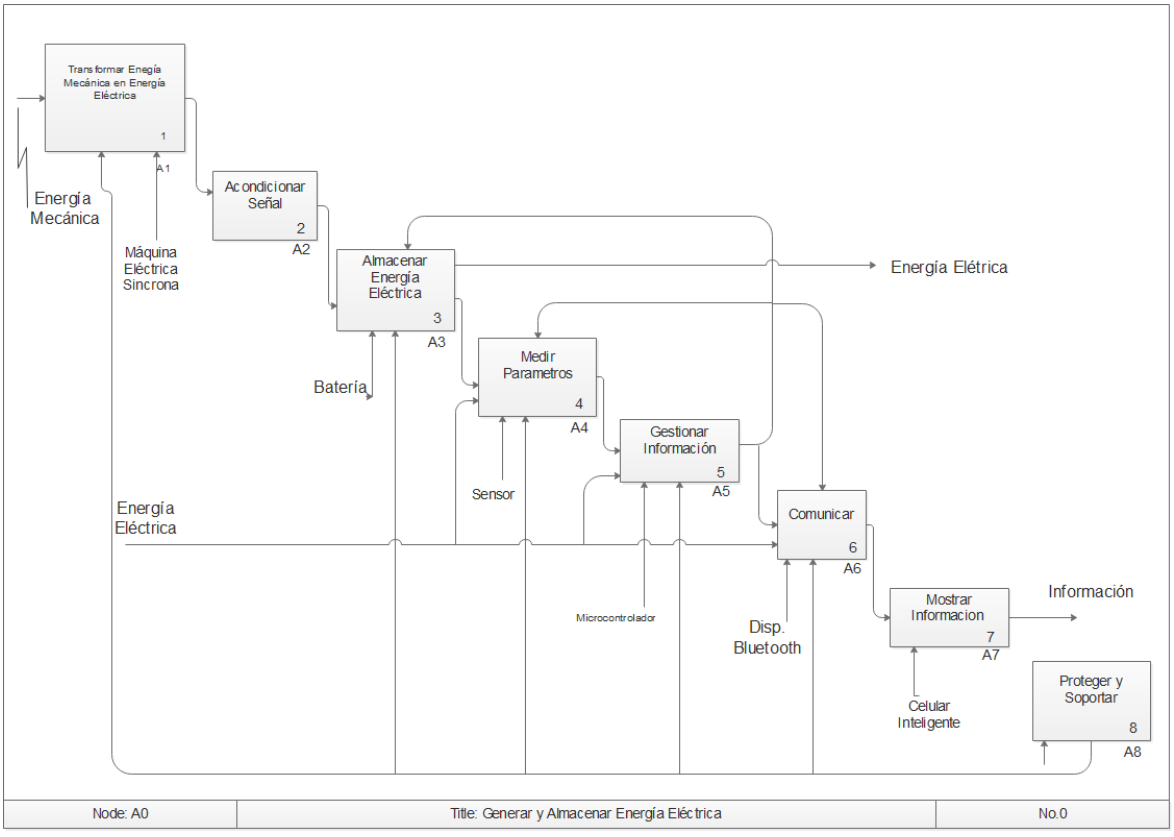


Figura 91. IDEF0 del sistema.

e.2. Sistemas y subsistemas

Desarrollando el diagrama IDEF0 para determinar las funciones más importantes y sus interacciones con el fin de darle una forma física al proyecto, se procedió a definir los sistemas y módulos que conforman la propuesta de solución. Para esto se utilizó la arquitectura física del sistema, que es una representación de cómo se transforman las funciones en sistemas, subsistemas y módulos, para unirlos y crear componentes físicos que convergen en un sistema con el fin de cumplir con sus requerimientos (figura 92).

***S*1 Sistema Central de Procesamiento de Datos:** Este sistema está constituido por el módulo **M5**, el cual se encarga gestionar las decisiones generales a partir de los datos que recibe, se encarga también

de preparar los datos para ser enviados, además de avisar si ocurre un error al cargar la batería. También se encuentra el módulo **M6** que se encarga principalmente de transmitir datos vía Bluetooth al dispositivo móvil del usuario.

***M*1 Módulo de energía:** Este módulo contiene al submódulo $\boldsymbol{sM_{11}}$, que se encarga transformar la energía mecánica en energía eléctrica por medio de una maquina síncrona. El sistema contiene también al submódulo $\boldsymbol{sM_{12}}$, el cual se encarga de acondicionar la señal para posteriormente ser almacenada en una batería.

***M*2 Módulo de almacenamiento de energía eléctrica:** Este módulo se encarga de almacenar la energía eléctrica en una batería portátil o en una batería de bicicleta eléctrica. Además de evitar una sobrecarga.

***M*3 Módulo de medición de tensión eléctrica:** Este módulo se encarga de monitorear la tensión eléctrica de los convertidores. Mide la tensión con la ayuda de un microcontrolador y su respectivo divisor de tensión. Esto permite medir la tensión de los convertidores para evitar mandar una sobrecarga a las baterías.

***M*4 Módulo de alimentación:** Este módulo se encarga de alimentar de energía eléctrica al sistema ***S*1**, y al módulo de ***M*3**, por medio de una batería de 9V. Alimenta al microcontrolador y parte de la electrónica de potencia.

***M*7 Modulo de interfaz máquina humano:** Este módulo se encarga de desplegar información como el valor de la tensión de los convertidores, tiempo del recorrido, distancia recorrida, velocidad promedio y ruta.

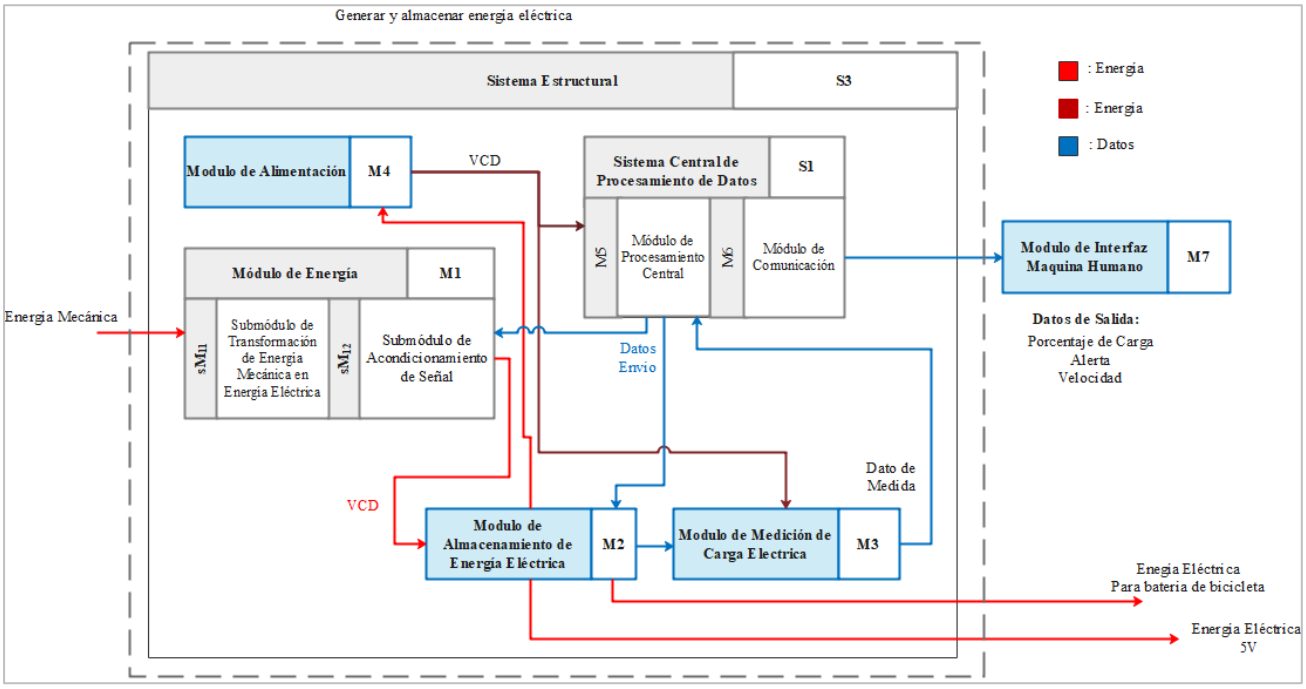


Figura 92. Arquitectura física del sistema.

f. Manufactura/implementación

A continuación, se describe el proceso a seguir para manufacturar la estructura mecánica y el circuito de la rueda frontal de bicicleta generadora de energía eléctrica.

f.1. Estructura mecánica

Las bandas trapezoidales se unirán al rin rodada 26 sin radios por medio de sikaflex 200+ y remaches de 30 mm que se inserten en los agujeros de los radios para fijarla en su lugar (figura 93).

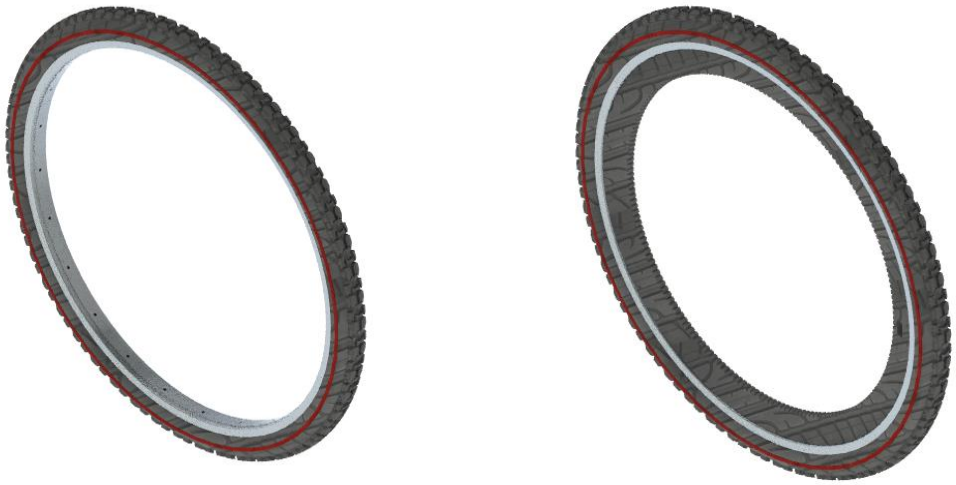


Figura 93. Ensamblaje de la rueda frontal de bicicleta con la banda trapezoidal C64.

Las poleas serán impresas en 3D usando PLA de fibra de carbono con una densidad de relleno del 80% para que sea muy resistente y aguante los 207.2942 N que se le aplica de forma horizontal debido al peso transmitido por los brazos de la horquilla hacia el buje de la estructura. Se verificará que exista un buen contacto entre la rueda lateral tipo polea y la correa trapezoidal para que exista fricción entre ellas (figura 94).

Figura 94. Ensamblaje de la banda trapezoidal C64 con las ruedas laterales tipo polea manufacturadas con plástico PLA por medio de una impresora 3D.

Se unirá el acoplamiento mecánico entre el motor y la banda trapezoidal C64 por medio de remaches aprovechando los 36 agujeros donde se colocan los radios en el motor MXUS-XF07 antes de integrarse al sistema, para ello se imprimirá el acoplamiento en una impresora 3D en dos partes con el propósito de unirlas posteriormente al motor y finalmente entre sí para colocarse de forma vertical en la rueda frontal, verificando que exista contacto entre el acoplamiento mecánico y la banda trapezoidal C64 (figura 95).

Figura 95. Ensamblaje acoplamiento entre el generador eléctrico y la banda trapezoidal con el motor MXUS-XF07.

Por separado se unirá el eje de rotación de la estructura con el eje de cierre rápido, se introducirá en el agujero interno del buje que tiene un diámetro interior de 5 mm el eje de cierre rápido que tiene un diámetro exterior de 4.5 mm Las placas laterales serán ensambladas al buje por medio de 6 remaches con diámetro de 5 mm y profundidad de 8 mm (figura 96).

El eje de cierre rápido se utiliza para que sin necesidad de utilizar ninguna herramienta se pueda colocar o quitar la rueda frontal de bicicleta generadora de energía eléctrica, para ello se debe sujetar un extremo y girar la palanca del otro, esto funciona porque el eje de cierre rápido cuenta con dos resortes internos de cada lado que le ayudan a sujetarse a la rueda de una forma sencilla.

Figura 96. Ensamblaje del eje de rotación con las tuercas de sujeción y el eje de cierre rápido.

Las placas paralelas de la estructura tienen un doblez para que puedan dejar libres los extremos del buje y que este pueda ajustarse a los brazos de la horquilla de la bicicleta, dichos brazos tienen una separación de 108 mm y una abertura de 10 mm, por lo que el eje de rotación tiene un radio de 12 mm y un corte de 10 mm de separación para que embone perfectamente, además está inclinado a 25° para que coincida con la horquilla y a su vez mantenga alineados verticalmente los dos motores. El doblez de las placas laterales está diseñado para que pueda sujetar los dos motores verticales que tienen un ancho de 108 mm (misma distancia de separación entre los brazos de la horquilla) y las ruedas laterales en forma de polea que tienen un ancho de 33 mm. Entre las ruedas tipo polea y las placas laterales existe una separación de 1 mm en cada lado para que no exista fricción durante la rotación de la rueda (figura 97).

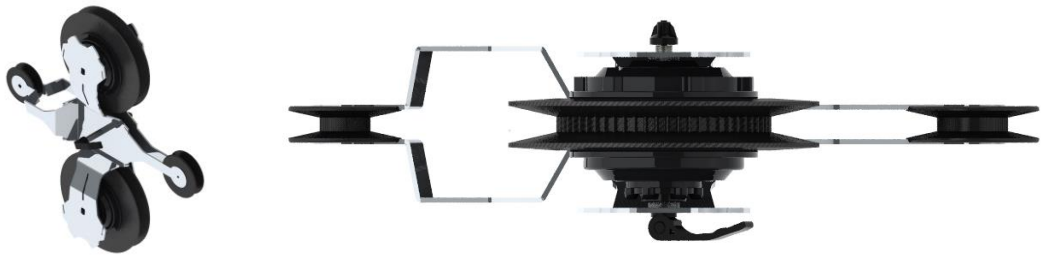


Figura 97. Ensamblaje de las placas laterales, el eje de rotación de cierre rápido, los generadores MXUS-XF07 y las ruedas laterales tipo polea.

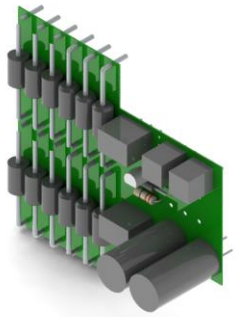
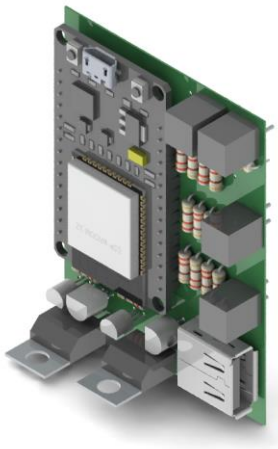
Finalmente se une el sistema con la rueda Kylin para que finalmente se pueda agregar a la bicicleta de pruebas Monk Kron con todo y el sistema de captación y acondicionamiento de energía eléctrica (figura 98).

Figura 98. Ensamblaje total de la rueda frontal de bicicleta generadora de energía eléctrica.

En la tabla 48 se muestra el listado de piezas a manufacturar, material y su forma de manufactura siguiendo el mismo orden descrito en la tabla 43:

Tabla 48. Lista de piezas a manufacturar para la rueda frontal generadora de energía eléctrica.

Nombre - Forma de manufactura - Material	Pieza por manufacturar
6.- Rueda lateral frontal tipo polea – Impresión 3D – Fibra de carbono plástico PLA.	
7.- Rueda lateral trasera tipo polea – Impresión 3D – Fibra de carbono plástico PLA.	
8.- Acoplamiento mecánico motor MXUS-XF07 y Banda trapezoidal C64 – Impresión 3D – Fibra de carbono plástico PLA.	
9.- Eje de rotación para el eje de cierre rápido – Torno y Fresadora – Aluminio 6061-T6.	

9.- Placa lateral derecha de la estructura mecánica – CNC de plasma y dobladora – Lámina de Aluminio 3003-H14	
10.- Placa lateral izquierda de la estructura mecánica – CNC de plasma y dobladora – Lámina de Aluminio 3003-H14	
15.- Sistema de captación y acondicionamiento de energía eléctrica – PCB 1: Acondicionamiento de energía eléctrica -	
16.- Sistema de captación y acondicionamiento de energía eléctrica – PCB 2: Comunicación y control inalámbrica -	

Vista explosionada del ensamblaje (figuras 99 y 100):

Figura 99. Vista explosionada del ensamblaje total de la rueda frontal de bicicleta generadora de energía eléctrica.

Figura 100. Vista explosionada del ensamblaje total de la rueda frontal de bicicleta generadora de energía eléctrica siendo montada en la bicicleta de pruebas Monk Kron.

f.2. Sistema de captación y acondicionamiento de energía eléctrica

Para el diseño de circuito, se diseño en dos partes ya que el espacio no permite tener una placa completa por lo tanto en la siguiente figura 101, se muestra una parte donde va la entrada de la señal a los diodos y se entrega la señal rectificada y suavisada por el capacitor para posteriormente pasar a los convertidores, ya sea el convertidor elevador o el convertidor reductor.

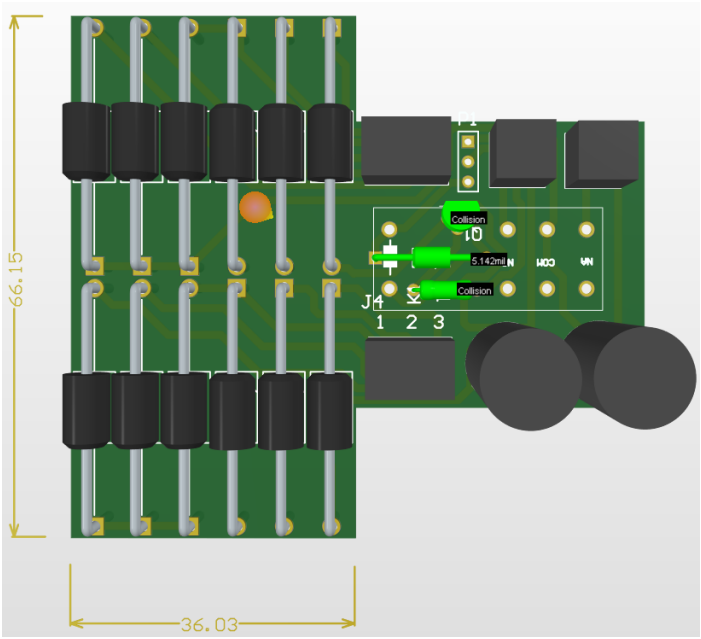


Figura 101. Primera placa pcb de rectificacion de señal.

FComo se ve en la figura 101, hay unos componentes de color verde, los cuales se encuentran dentro del lugar del relvador, pero por las dimensiones del relevador, no puede permanecer en ese lugar, por lo tanto, se aprovecho el lugar para poner los elementos de color verde, en este caso son el transistor, el diodo de descarga y la resistencia.

Para la siguiente placa se tiene la parte de control por parte de la ESP32, y la parte de potencia donde como compuertas y además de la lectura de las tensiones de salida por parte de los convertidores, ver figura 102.

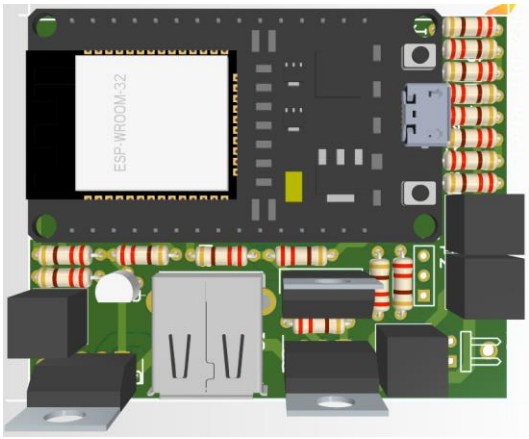


Figura 102. Circuito PCB de potencia y de control de generador de enegía de bicicleta.

Para la contruccion de las placas, se mandarán a fabricar, los modulo se va a construir de una sola capa, por el método de serigrafia ya que la placa se encontrará en ambientes corrosivos, por lo cual el método de serigrafia, brinda una mayor protección a las pistas y a sus componentes.

Se utilizará un taladro para realizar perforaciones en la placa con diferentes medidas de brocas. Se limpiará la placa con alcohol isopropílico para eliminar la grasa existente y posteriormente se soldarán los componentes correspondientes.

Tomese en cuenta que microcontrolador es alimentado en la entrada TB5 con tensiones de 5 a 15V, gracias a que tiene un regulador integrado ASM1117.

PRUEBAS

a. Propios del desarrollo tecnológico e investigación

A continuación, se desarrollan pruebas de la parte mecánica, electrónica y de la aplicación móvil en el sistema operativo Android del sistema para comprobar que se hayan alcanzado los objetivos planteados.

a.1. Simulaciones mecánicas

Tras haber elegido un espesor de 20.2 mm y redondeo de 4 mm para el buje del sistema y un espesor de 3.4 mm para la placa lateral de la estructura mecánica, se analizará la deformación que podrán tener las piezas a manufacturar y además se hará un análisis de vibraciones para encontrar las frecuencias de resonancia en cada una. Se empieza el análisis de la estructura mecánica simulando los esfuerzos de Von Mises y el Esfuerzo principal máximo en el buje (figuras 103 a 107).

Buje del sistema

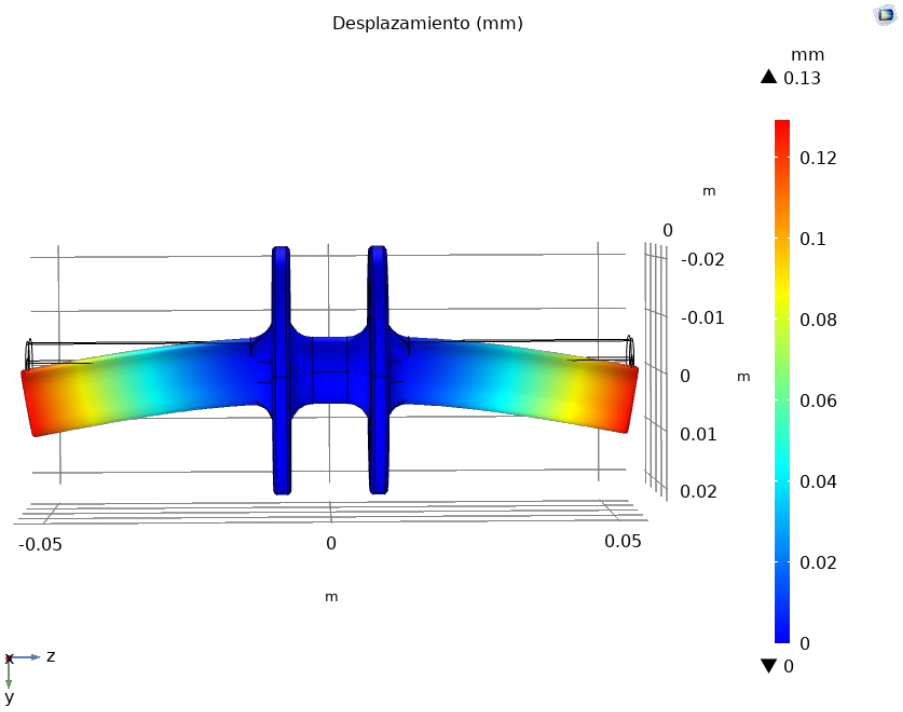


Figura 103. Deformación mecánica en la zona elástica - Buje con ancho entre placas de 35 mm y redondeo de 4 mm.

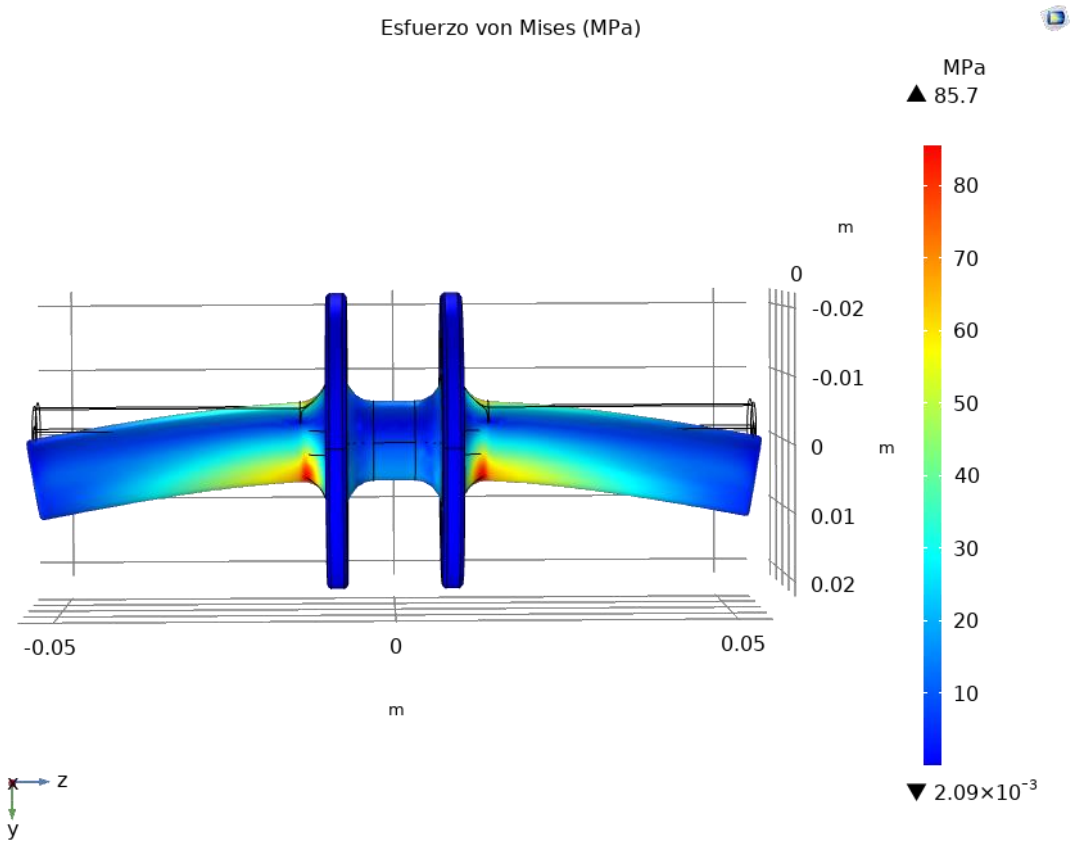


Figura 104. Esfuerzo de Von Mises Buje con ancho entre placas de 35 mm y redondeo de 4 mm.

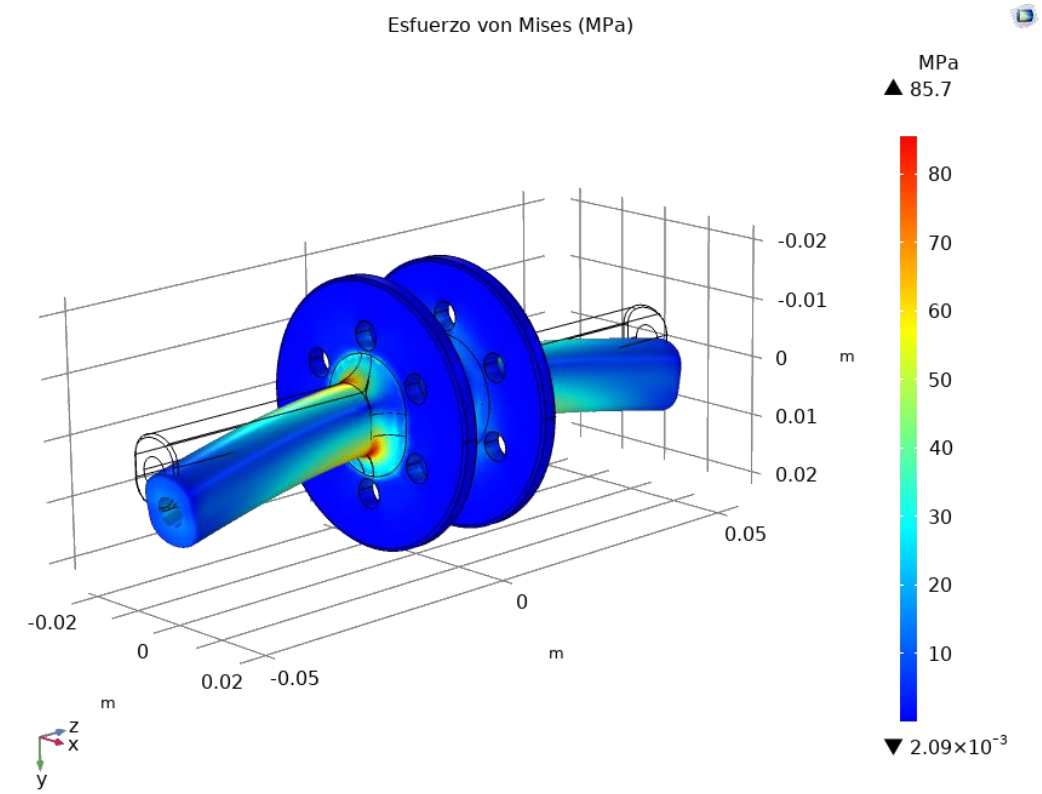


Figura 105. Esfuerzo de Von Mises - Buje con ancho entre placas de 35 mm y redondeo de 4 mm vista isométrica.

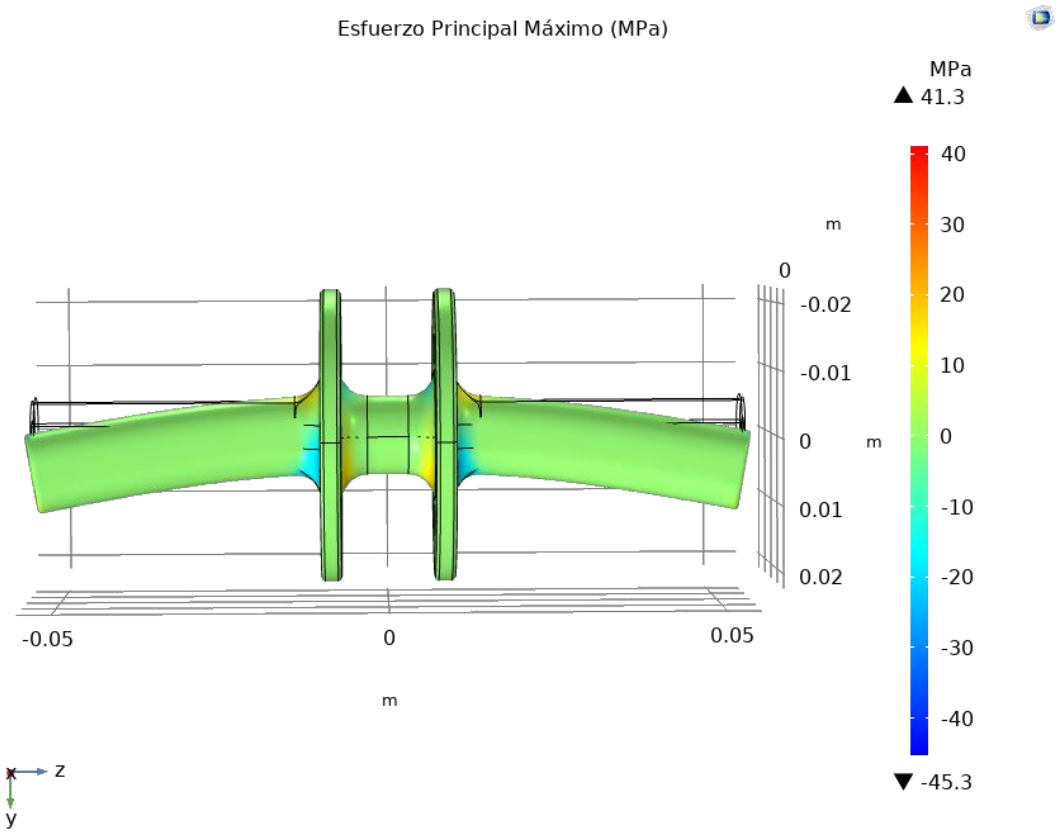


Figura 106. Esfuerzo Principal máximo - Buje con ancho entre placas de 35 mm y redondeo de 4 mm.

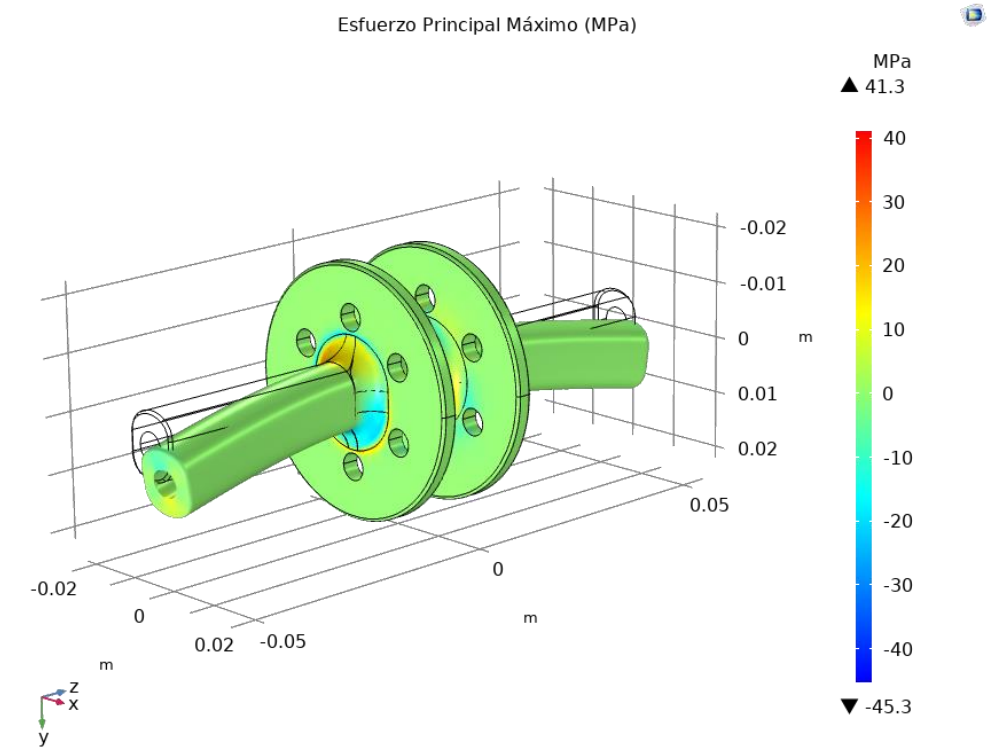


Figura 107. Esfuerzo Principal máximo - Buje con ancho entre placas de 35 mm y redondeo de 4 mm vista isométrica.

Al efectuar el análisis de armónicos se puede notar que la frecuencia de resonancia menor es muy alta, debido a que el sistema será de uso urbano y la bicicleta transitará a velocidades muy bajas, aproximadamente de 15 a 20 km/h, no hay peligro de falla mecánica causada por vibraciones en el buje (figuras 108 a 113).

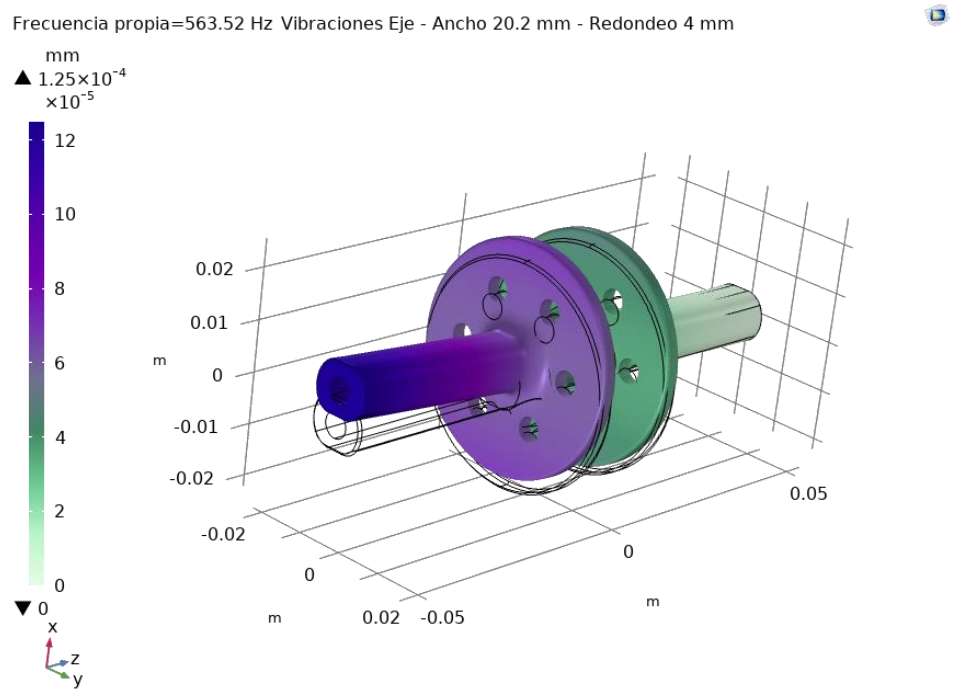


Figura 108. Frecuencia de resonancia - Buje con ancho entre placas de 35 mm y redondeo de 4 mm - armónico 1: 563.52 Hz.

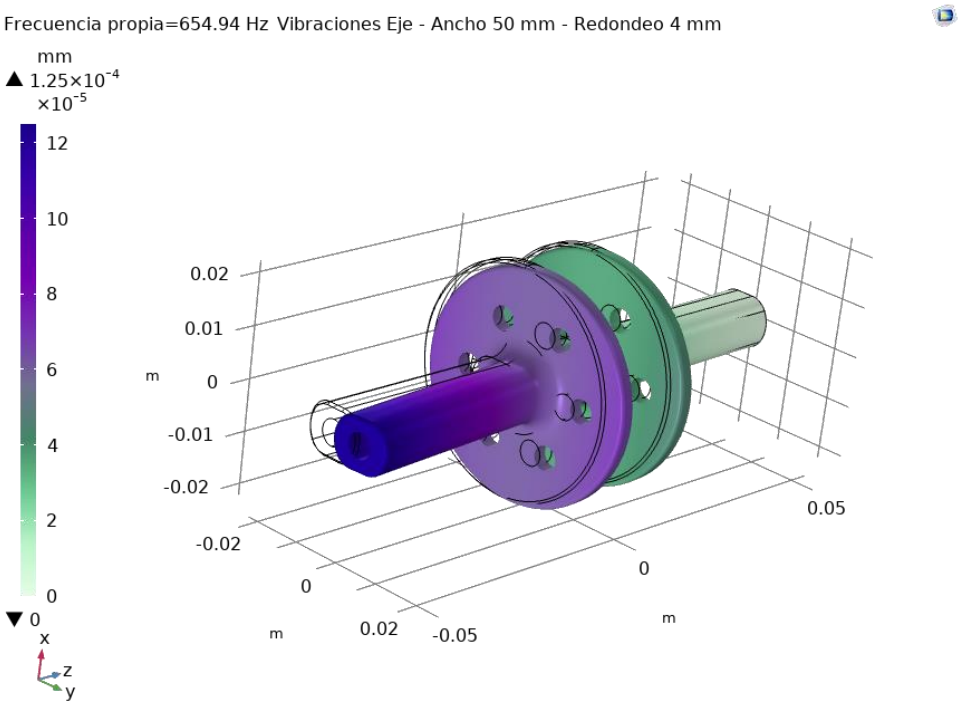


Figura 109. Frecuencia de resonancia - Buje con ancho entre placas de 35 mm y redondeo de 4 mm - armónico 2: 659.94 Hz.

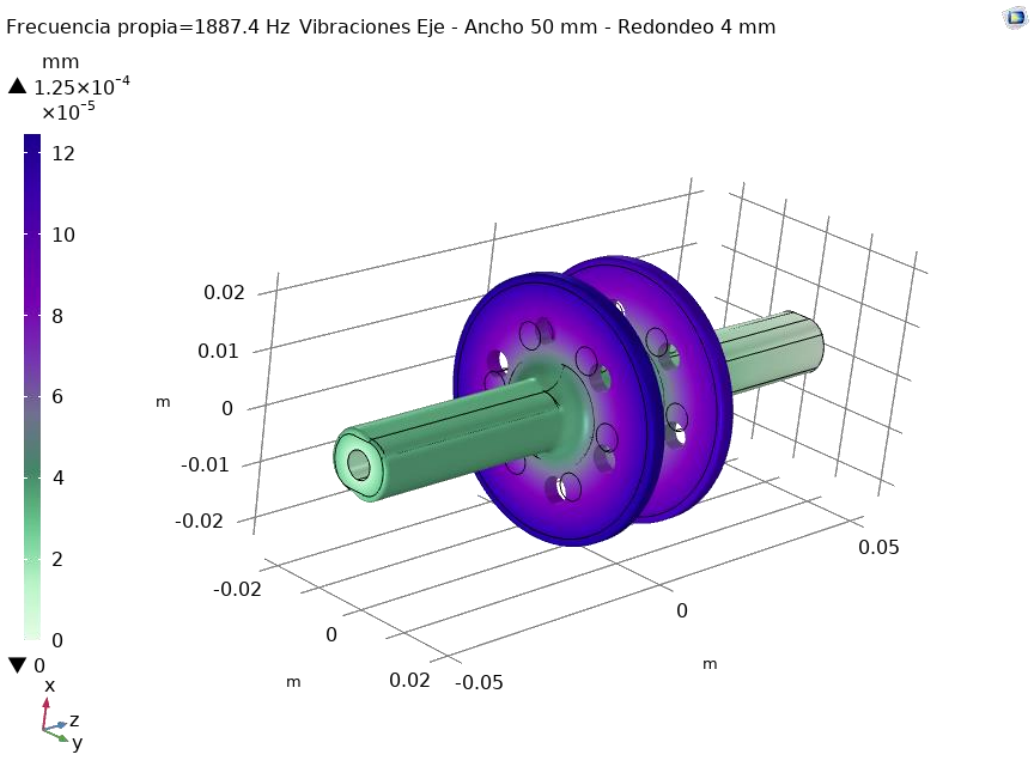


Figura 110. Frecuencia de resonancia - Buje con ancho entre placas de 35 mm y redondeo de 4 mm - armónico 3: 1887.4 Hz.

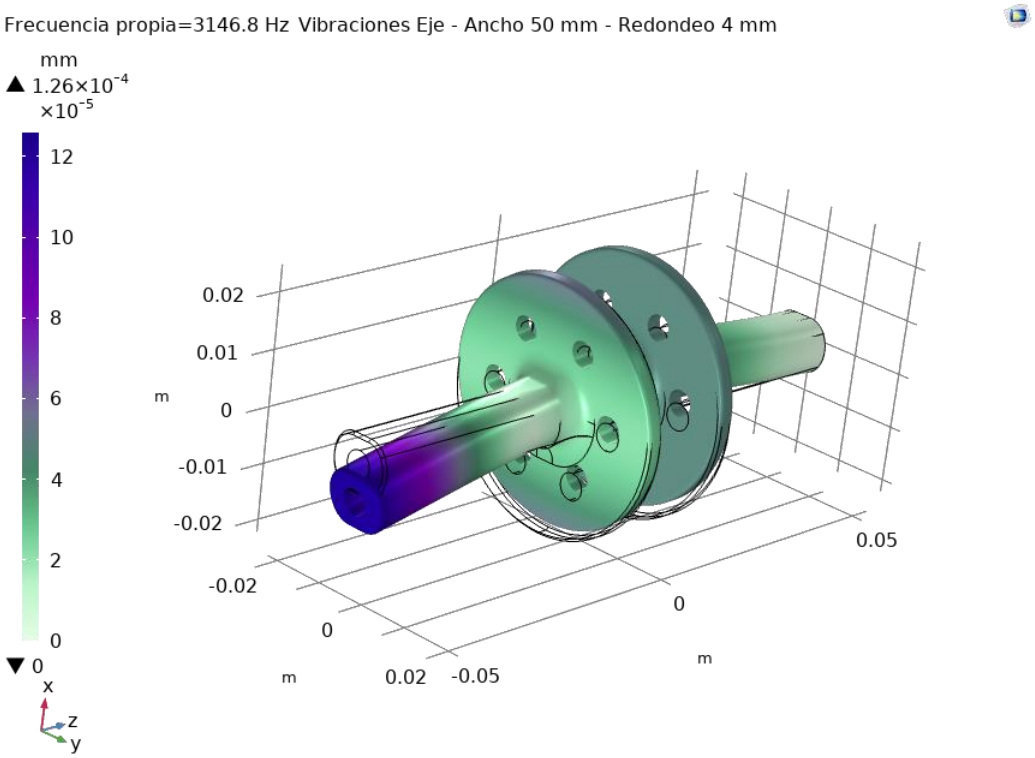


Figura 111. Frecuencia de resonancia - Buje con ancho entre placas de 35 mm y redondeo de 4 mm - armónico 4: 3146.8 Hz.

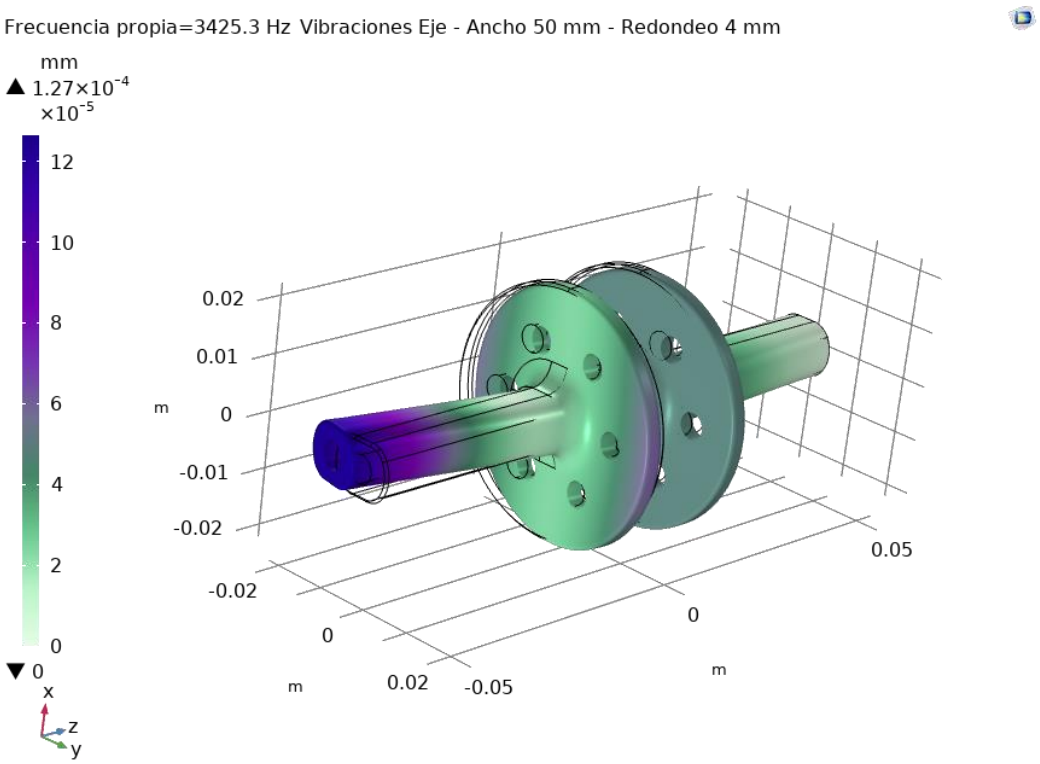


Figura 112. Frecuencia de resonancia - Buje con ancho entre placas de 35 mm y redondeo de 4 mm - armónico 5: 3425.3 Hz.

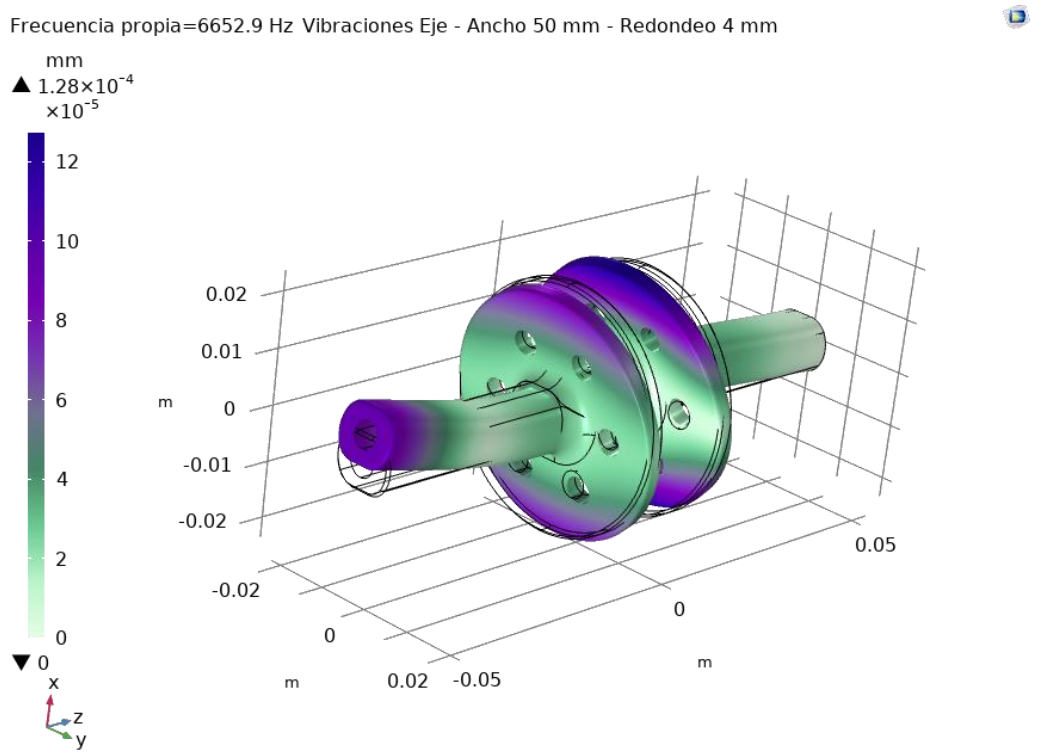


Figura 113. Frecuencia de resonancia - Buje con ancho entre placas de 35 mm y redondeo de 4 mm - armónico 6: 6652.9 Hz.

Placa lateral

Se repite el mismo análisis de deformación, esfuerzo y vibraciones hecho al buje de la estructura, pero ahora aplicado a la placa lateral, empezando por realizar las simulaciones de esfuerzos de Von Mises y el Esfuerzo principal máximo (figuras 114, 115, 116 y 117).

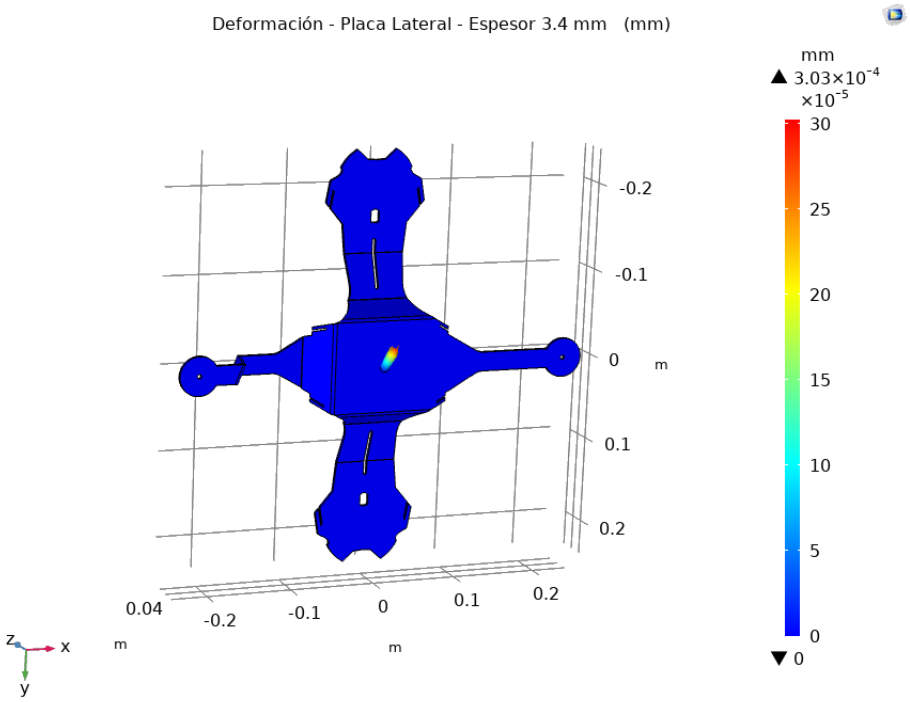


Figura 114. Deformación mecánica en la zona elástica – Placa lateral con espesor de 3.4 mm.

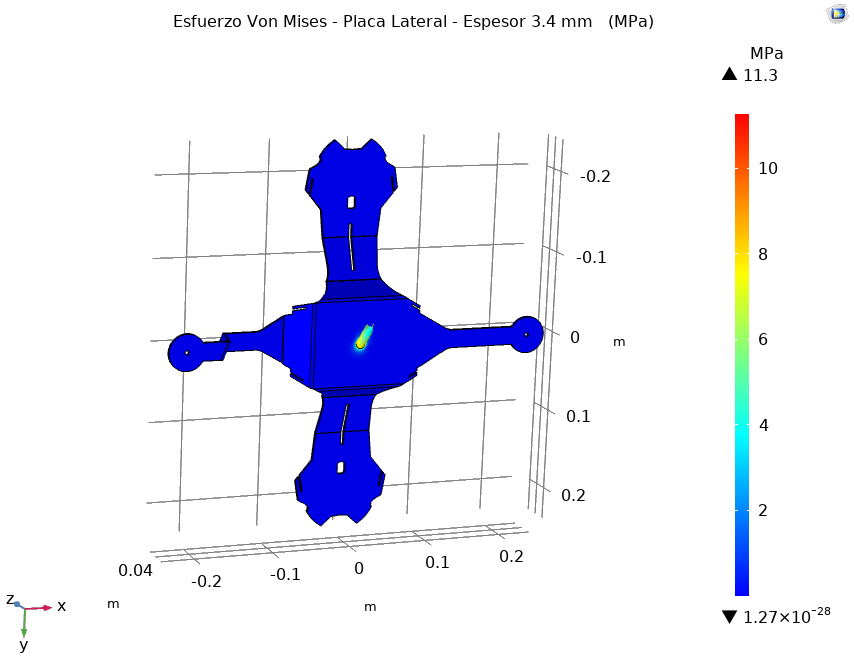


Figura 115. Esfuerzo de Von Mises – Placa lateral con espesor de 3.4 mm.

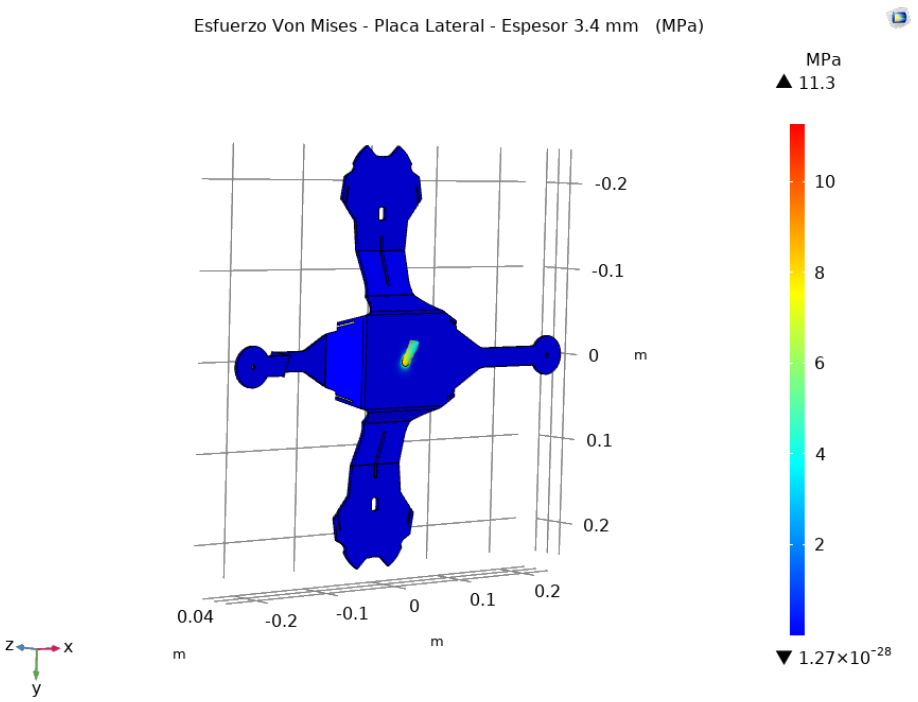


Figura 116. Esfuerzo de Von Mises – Placa lateral con espesor de 3.4 mm vista isométrica.

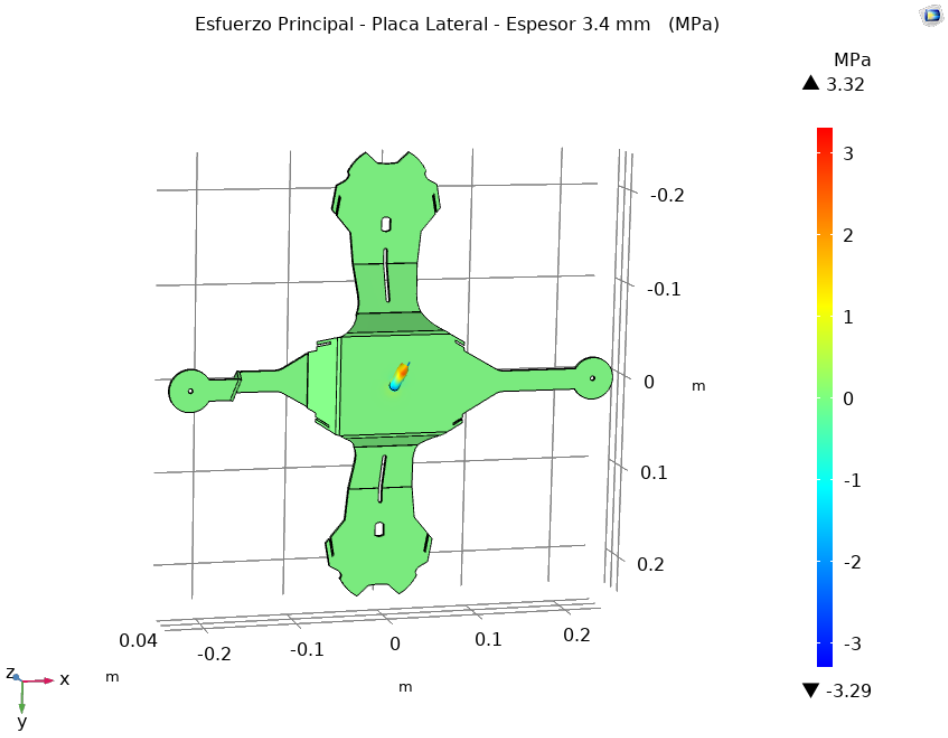


Figura 117. Esfuerzo Principal máximo – Placa lateral con espesor de 3.4 mm.

Al efectuar el análisis de armónicos se puede notar que la frecuencia de resonancia menor es baja, por lo que podría haber peligro de falla debido a las oscilaciones violentas de las zonas vistas en la simulación, por lo que es importante que en estos puntos haya soportes que resistan las oscilaciones, los soportes serán manufacturados por medio de una impresora 3D usando filamento de fibra de carbono,

no obstante, el peligro de falla en las placas laterales es bajo porque la mayoría de las oscilaciones son absorbidas por el buje y además la velocidad de operación es muy baja (figuras 118 a 123).

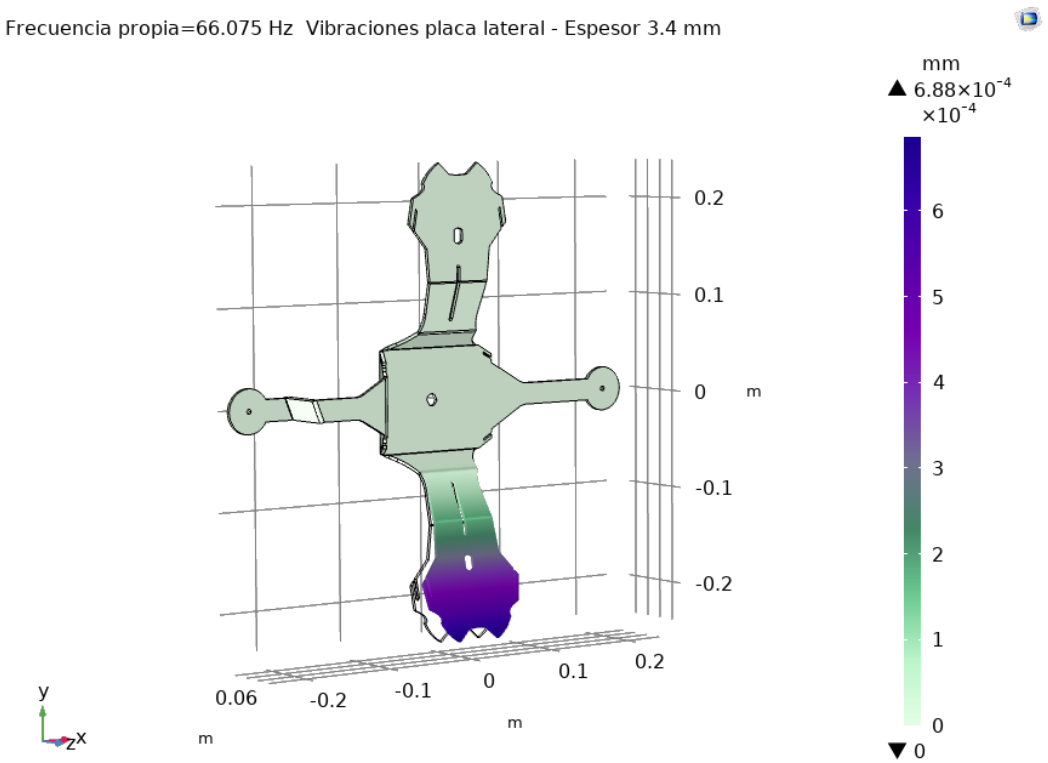


Figura 118. Frecuencia de resonancia – Placa lateral con espesor de 3.4 mm - armónico 1: 66.075 Hz.

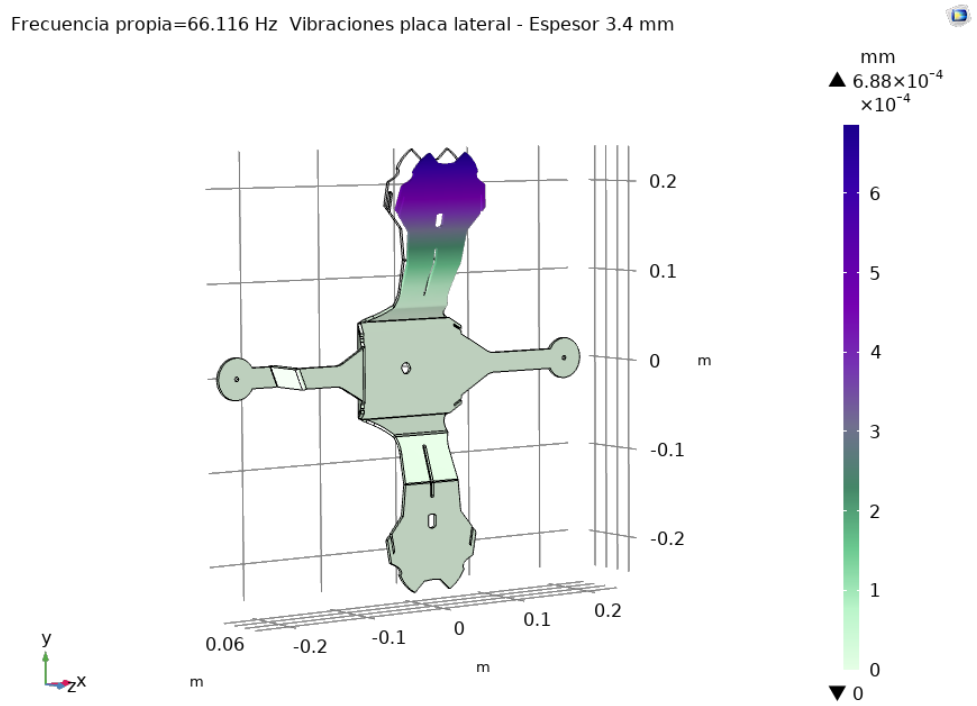


Figura 119. Frecuencia de resonancia – Placa lateral con espesor de 3.4 mm - armónico 2: 66.116 Hz.

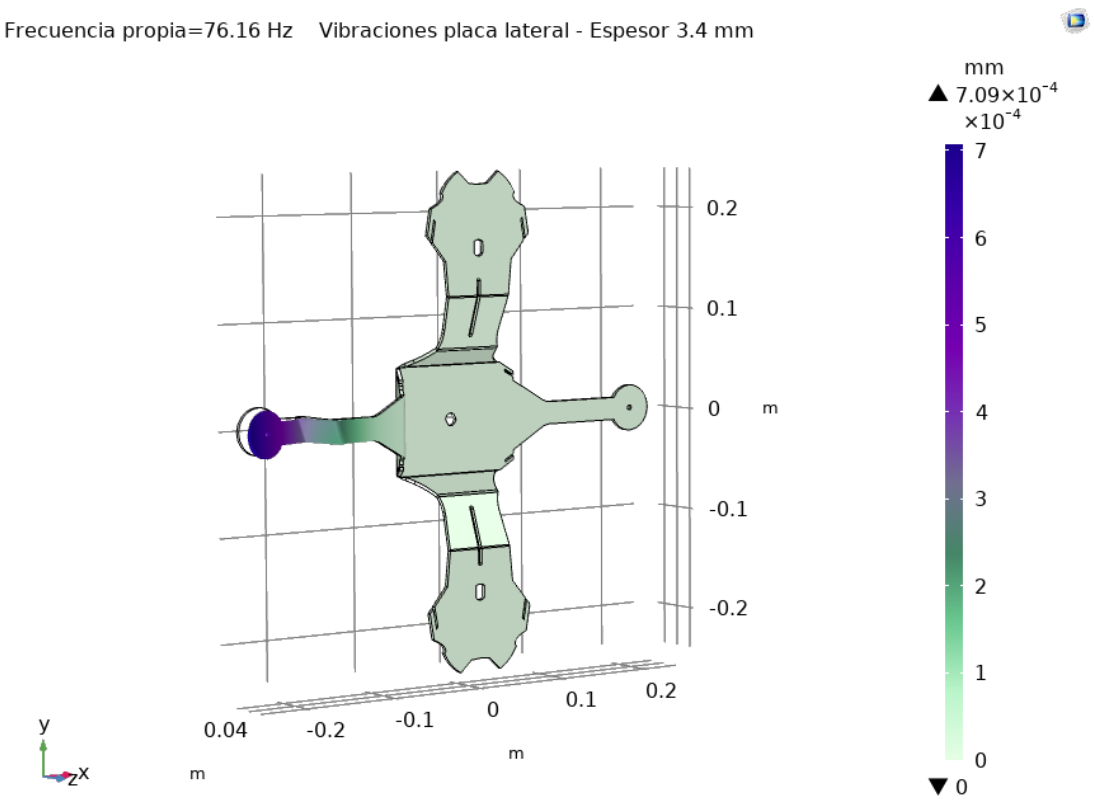


Figura 120. Frecuencia de resonancia – Placa lateral con espesor de 3.4 mm - armónico 3: 71.16 Hz.

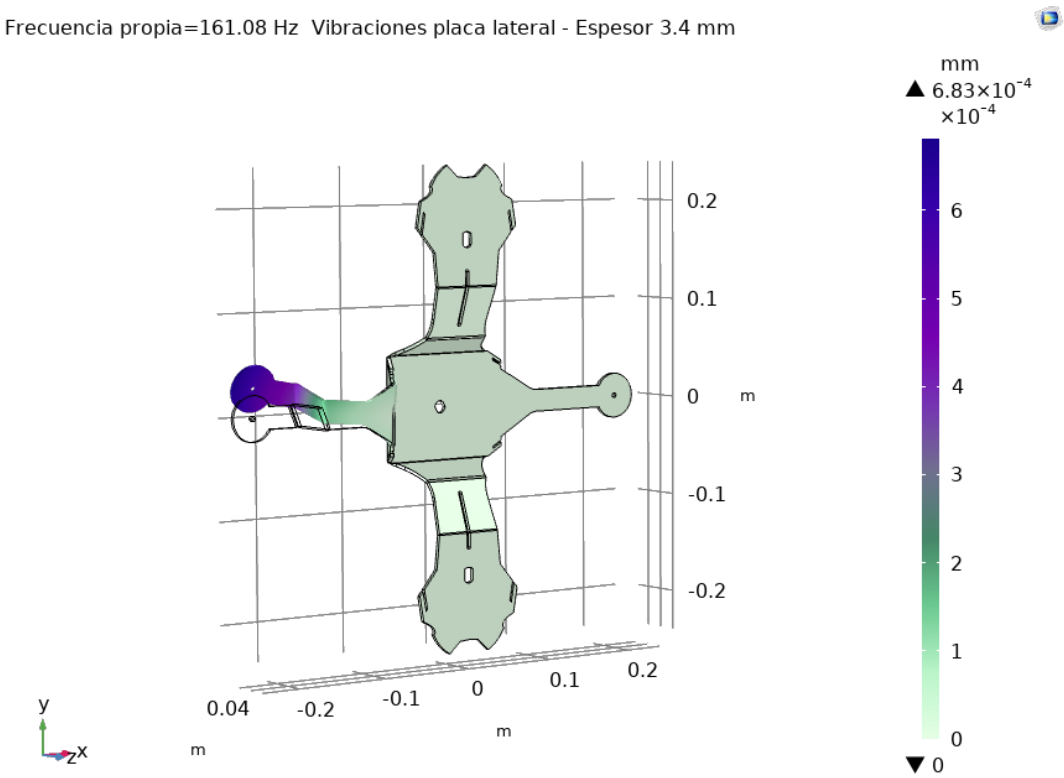


Figura 121. Frecuencia de resonancia – Placa lateral con espesor de 3.4 mm - armónico 4: 161.08 Hz.

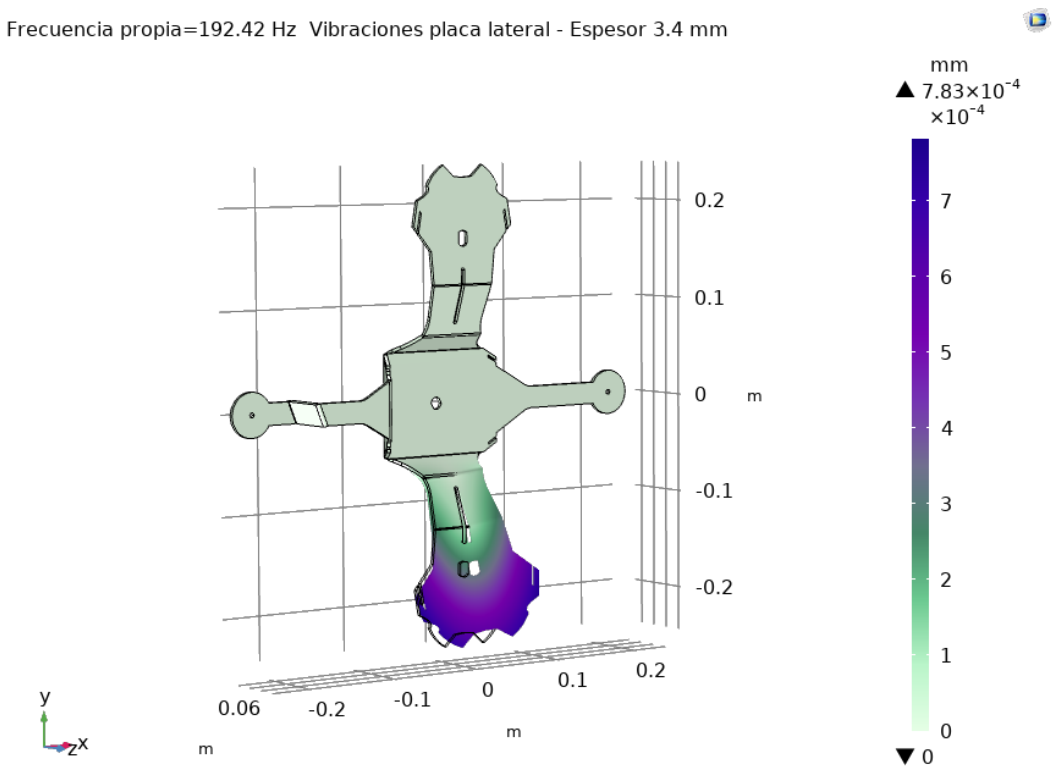


Figura 122. Frecuencia de resonancia – Placa lateral con espesor de 3.4 mm - armónico 5: 192.42 Hz.

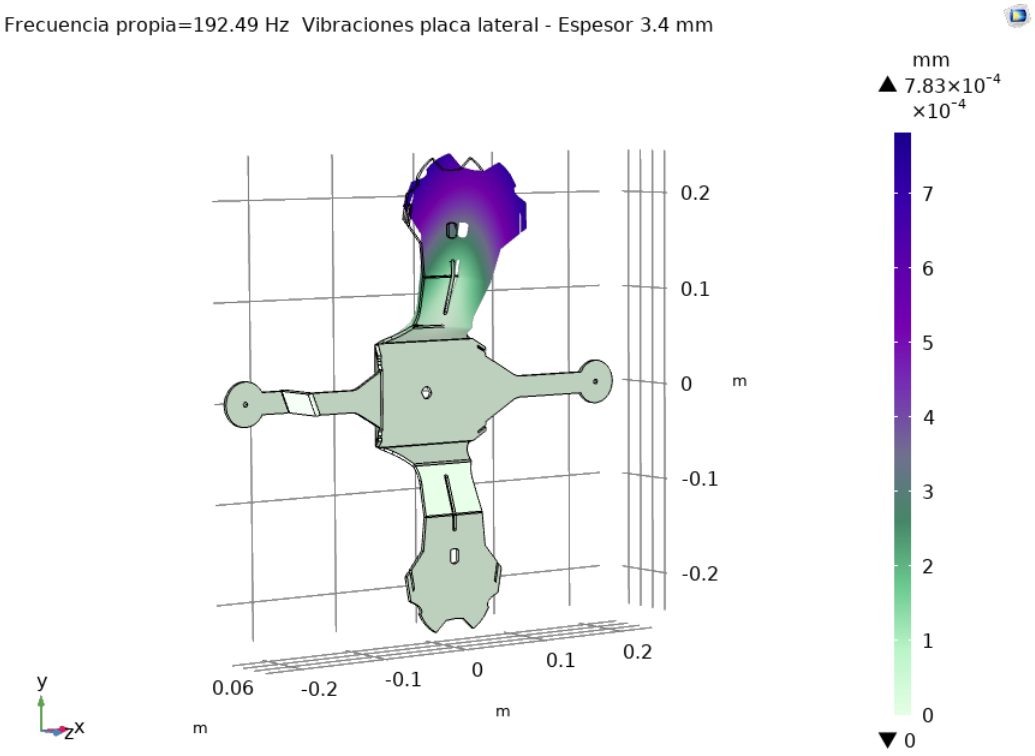


Figura 123. Frecuencia de resonancia – Placa lateral con espesor de 3.4 mm - armónico 6: 192.49 Hz.

Al realizar las pruebas de la rueda real se comprobó que los generadores giraban mejor si se encontraban inclinadas a 45°, a continuación, se harán pruebas de la estructura con una inclinación inicial de 45° y posteriormente con otros ángulos ya que el ángulo de inclinación de la horquilla varía dependiendo de la bicicleta.

Fmax = 490.5 N
Wm = 14.715 N

A partir de los datos mencionado y mostrados en la figura se procederá a encontrar los valores de N1, N2 y F.

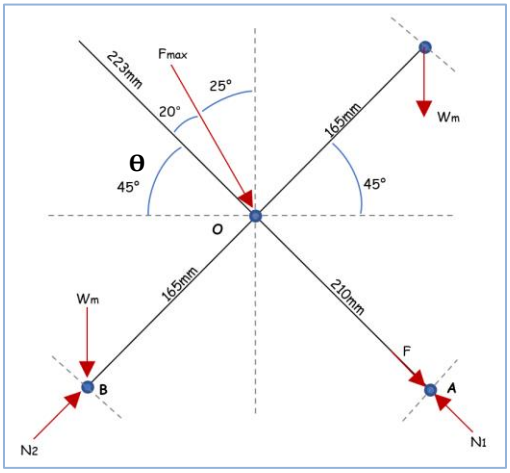


Figura 124. Análisis de la estructura con diferentes ángulos de inclinación.

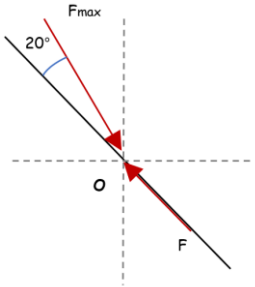
Se calcula la sumatoria de momentos en A para encontrar el valor de N2.

$\Sigma M_A = 210mm * W_m Cos45 - 165mm * W_m Sin45 + 210mm * F_{max} Sin20 - 210mm * N_2 + 210mm * W_m Cos45 + 165mm * W_m Sin45 = 0$…ecuación 279

$420mm * W_m Cos45 + 210mm * F_{max} Sin20 - 210mm * N_2 = 0$…ecuación 280

$N_2 = \frac{4370.132\ N*mm + 35229.784\ N*mm}{210mm} = 188.571\ N$…ecuación 281

Análisis de fuerzas en O:

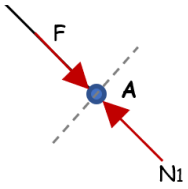


$F = F_{max} Cos20$ …ecuación 282

$F = 490.5\ N * Cos20$ …ecuación 283

$F = 460.919\ N$ …ecuación 284

Análisis de fuerzas en A



$F = N_1$ …ecuación 285
$N_1 = 460.919\ N$ …ecuación 286

En la siguiente table se muestran

Tabla 49. Comparación con diferentes ángulos de inclinacio (θ) de la estructura.

Θ	**N1 (N)**	**N2 (N)**	**F (N)**
45°	460.919	188.571	460.919
60°	401.794	306.826	401.794
70°	346.835	374.491	346.835
80°	281.339	430.776	281.339

Tomando como valores máximos de N1 y N2 calculamos el ancho (a) mínimo de la placa de la estructura de aluminio serie 3003-H14.

Para N1:
El grosor de la placa es de s = 3.4 mm = 3.4x10^{-3}m y con un esfuerzo de cedencia de 82.7371 MPa.

$P = \frac{F_{max}}{A_{min}}$…ecuación 287

$A_{min} = \frac{F_{max}}{\sigma_t}$…ecuación 288

$s * a_{min} = \frac{F_{max}}{\sigma_t}$…ecuación 289

$a_{min} = \frac{F_{max}}{s\sigma_t} = \frac{460.919\ \text{N}}{3.4x10^{-3}m*82.7371\ MPa}$…ecuación 290

$a_{min} = 1.63x10^{-3}m = 1.63\ mm$…ecuación 291

Para N2:
El grosor de la placa es de s = 3.4 mm = 3.4x10^{-3}m y con un esfuerzo de cedencia de 82.7371 MPa.

$P = \frac{F_{max}}{A_{min}}$…ecuación 292

$A_{min} = \frac{F_{max}}{\sigma_t}$…ecuación 293

$s * a_{min} = \frac{F_{max}}{\sigma_t}$…ecuación 294

$a_{min} = \frac{F_{max}}{s\sigma_t} = \frac{430.776\ \text{N}}{3.4x10^{-3}m*82.7371\ MPa}$…ecuación 295

$a_{min} = 1.53x10^{-3}m = 1.53\ mm$…ecuación 296

Como se puede observar en las ecuaciones 291 y 296, esos valores del ancho son los mínimos que la placa pueda tener antes de que tenga una roptura.

a.2. Simulación de la aplicación Android

En la siguiente imagen muestra como están unidos los fragmentos dentro de la aplicación. Empezando por inicioFragment que avisará al usuario que debe tener el bluetooth encendido para poder utilizar la aplicación. Después lo lleva al siguiente fragmento que es seleccionFragment en donde se va a decidir que dispositivo suministrar energía eléctrica, si una batería portátil o una bicicleta eléctrica. En setupFragment pedirá al usuario que ingrese su nombre y peso. En runFragment es el apartado en donde se monitorea el recorrido del usuario. Después de terminar un recorrido, en trackingFragment se guarda los recorridos realizados por el usuario, así como la velocidad, distancia recorrida, calorías quemadas y el tiempo del trayecto. Tambien tenemos dos fragmentos que se encuentran de forma paralela al runFragment, los cuales son settingsFragment y statisticsFragment. En settingsFragments se podrá cambiar los datos del usuario y en statisticsFragment se mostrará un grafíca con el rendimiento del usuario.

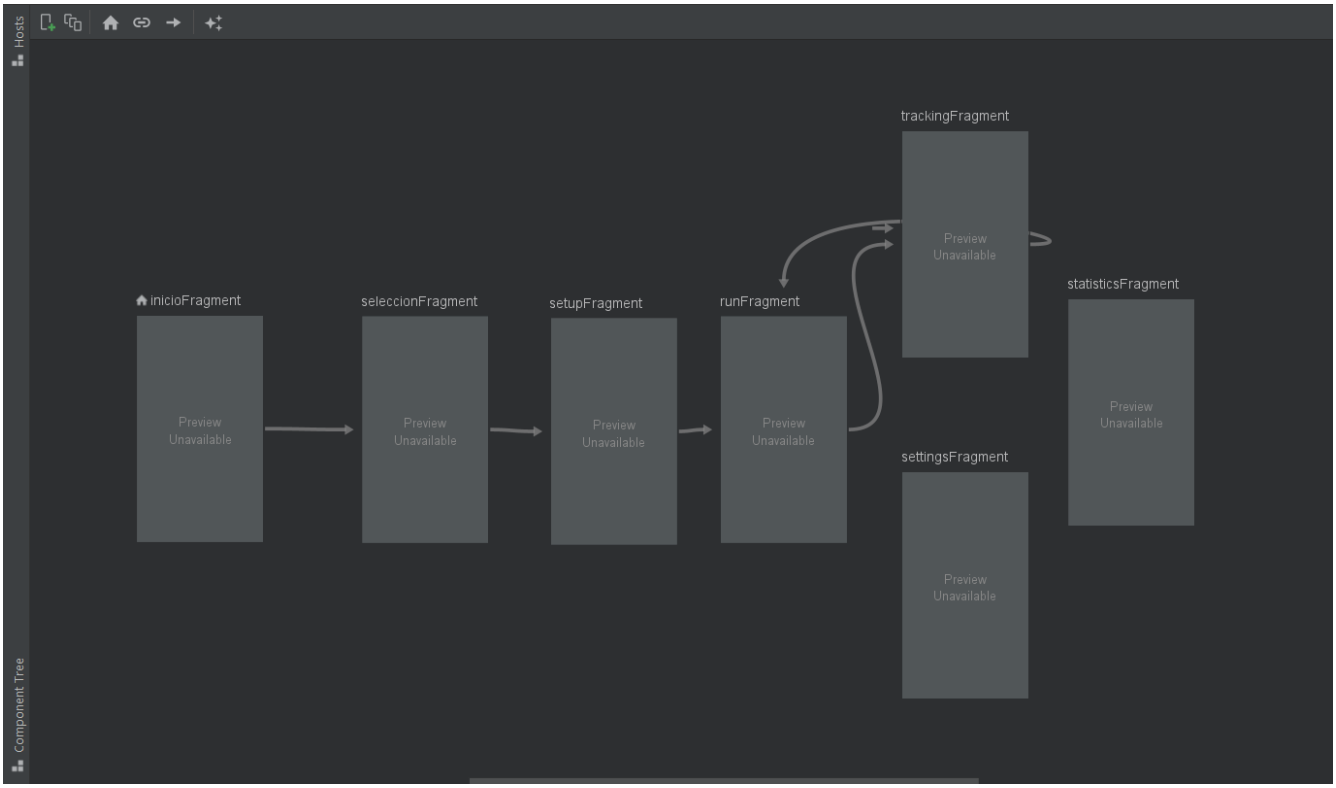


Figura 125. Fragmentos de la aplicación.

De la figura 126 a la 151 se puede observar el funcionamiento de la aplicación Android, en un inicio el usuario debe introducir su nombre y peso, después la aplicación se dirigirá a la ventana de historial, en donde al dar clic al símbolo de + se iniciará un nuevo recorrido, en esta nueva pantalla se mostrará el tiempo transcurrido y la ruta, al acabar un recorrido, en la ventana de rendimiento aparecerá la distancia, velocidad promedio, fecha, hora, ruta, calorías quemadas y duración de cada recorrido.

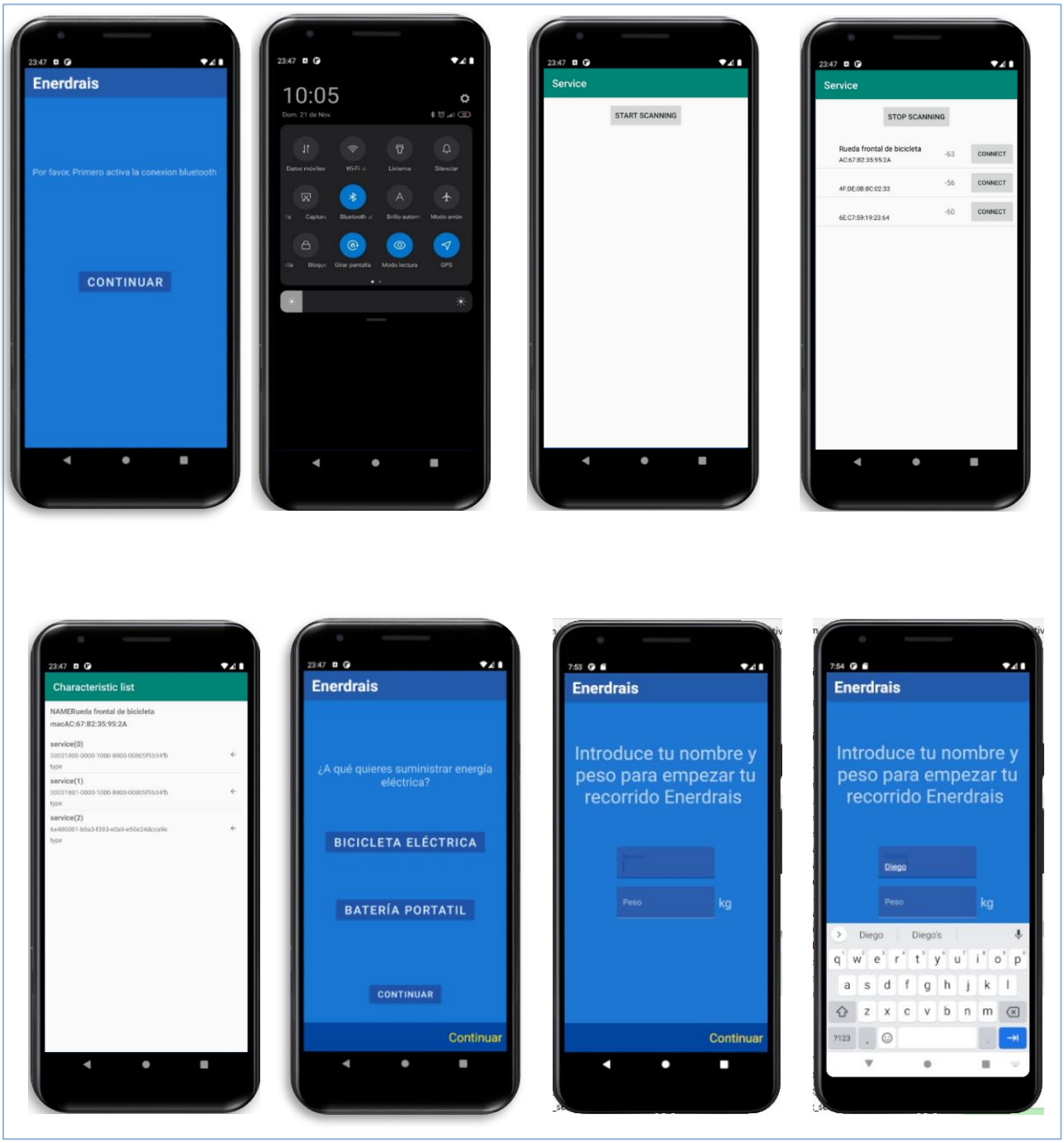


Figura 126. Capturas del funcionamiento de la aplicación 1.

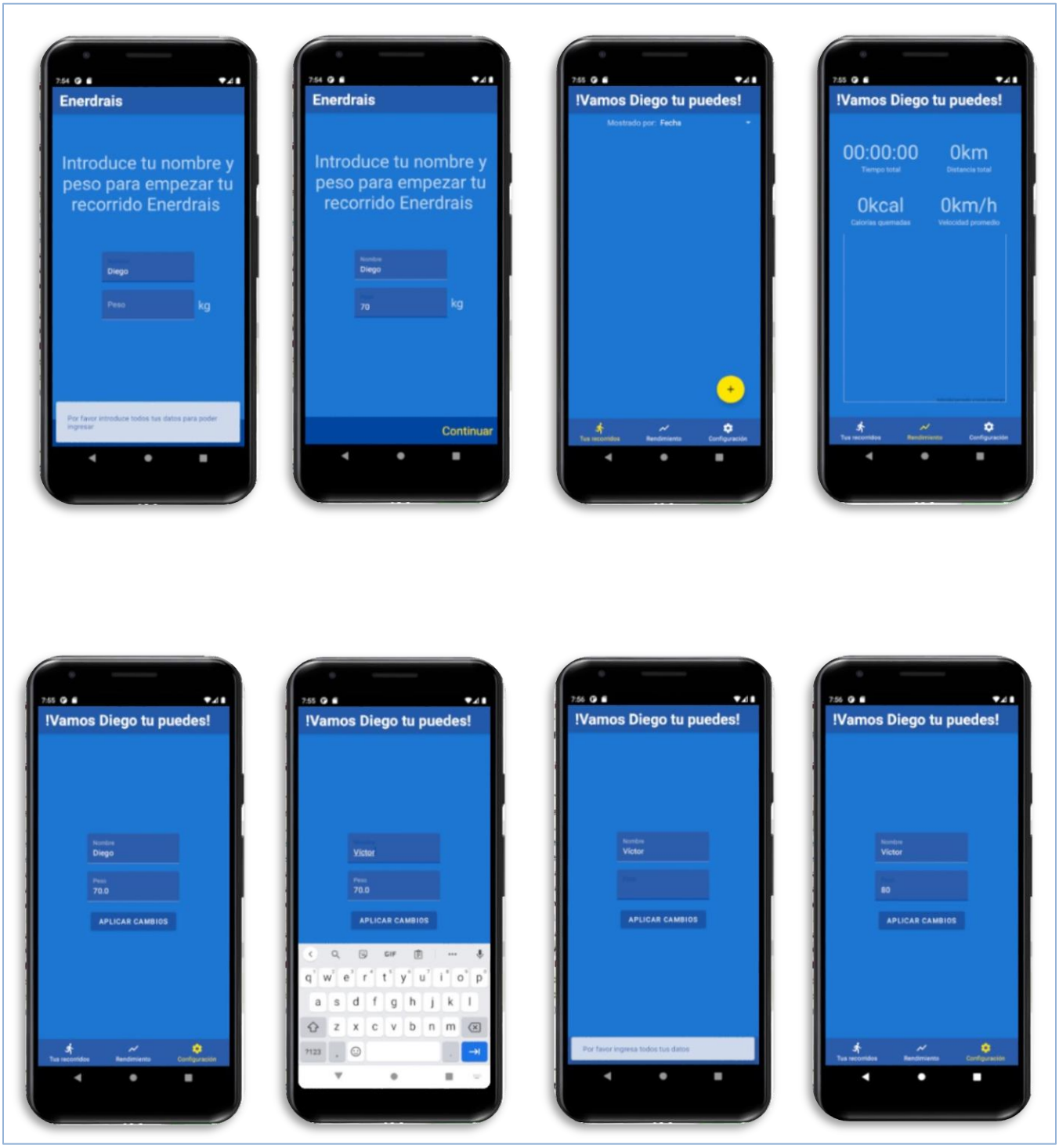


Figura 127. Capturas del funcionamiento de la aplicación 2.

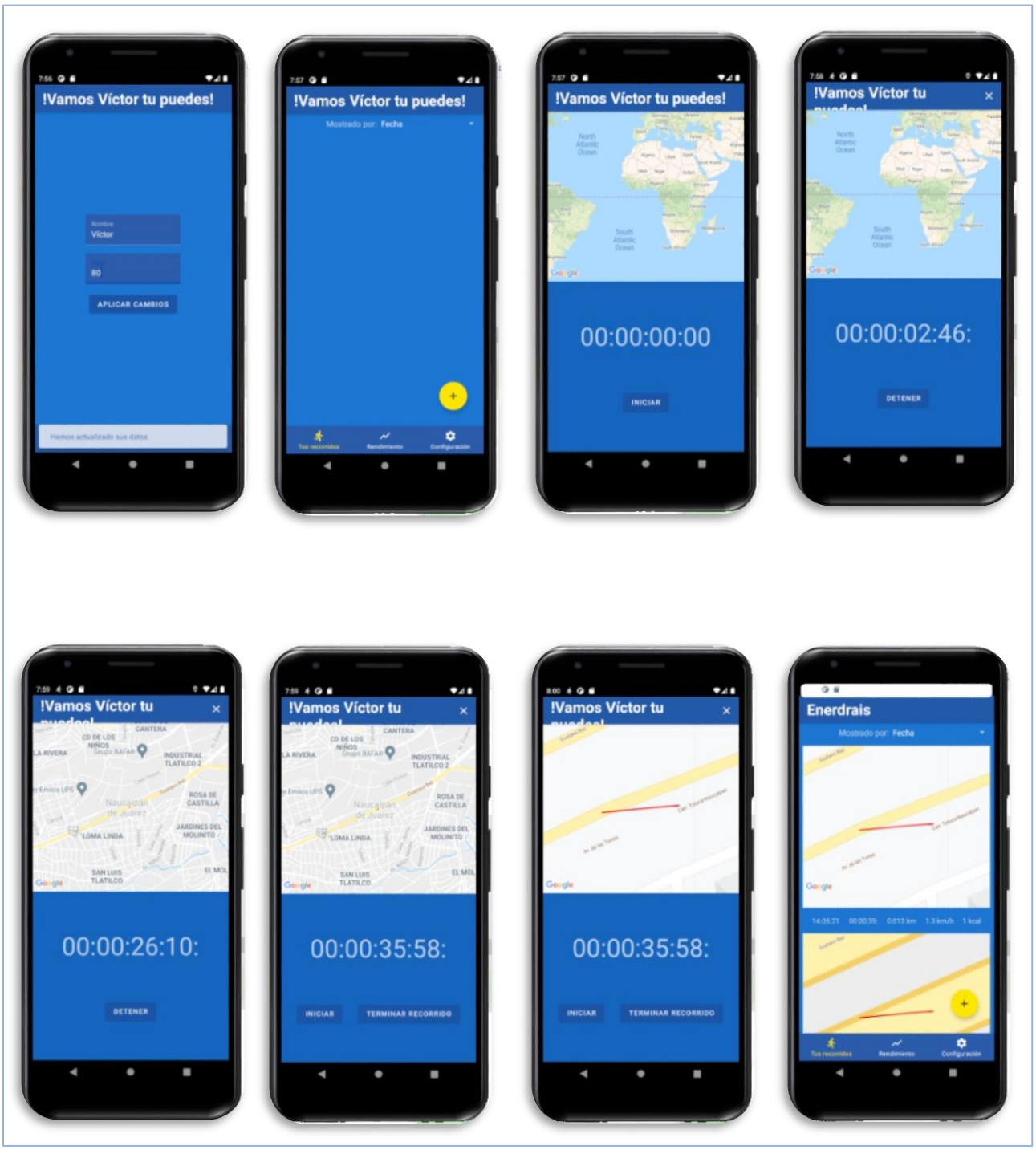


Figura 128. Capturas del funcionamiento de la aplicación 3.

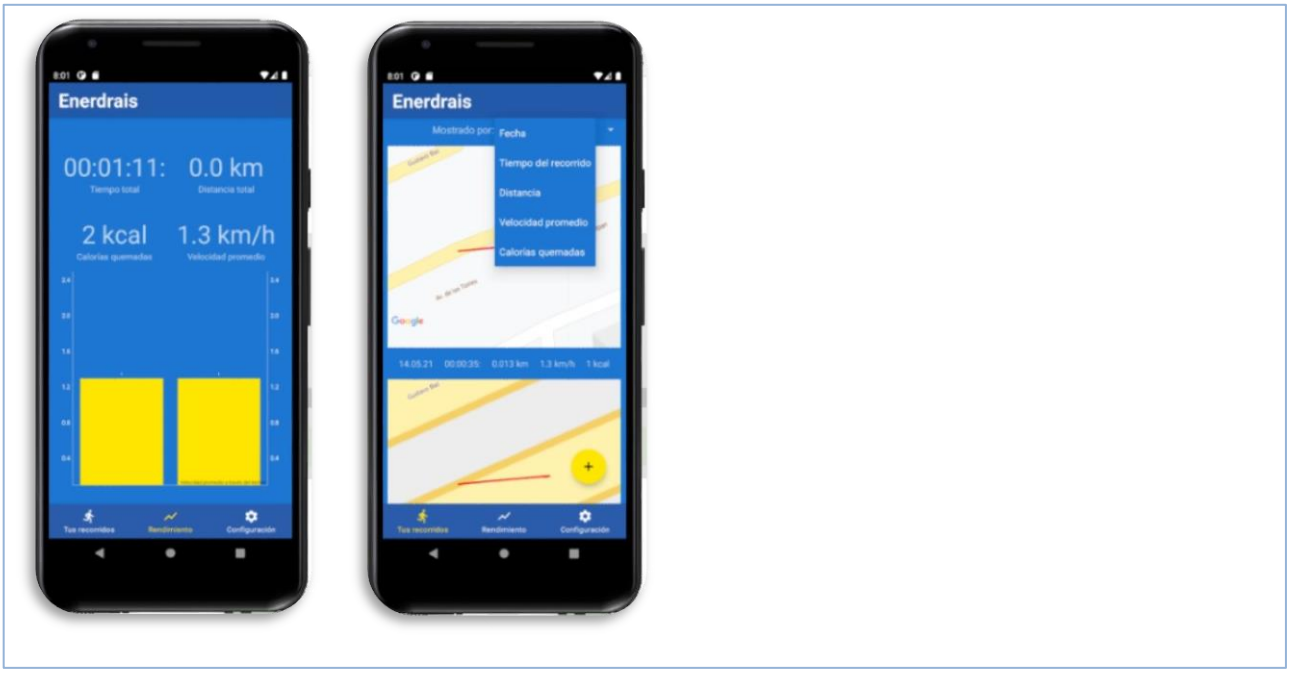


Figura 129. Capturas del funcionamiento de la aplicación 4.

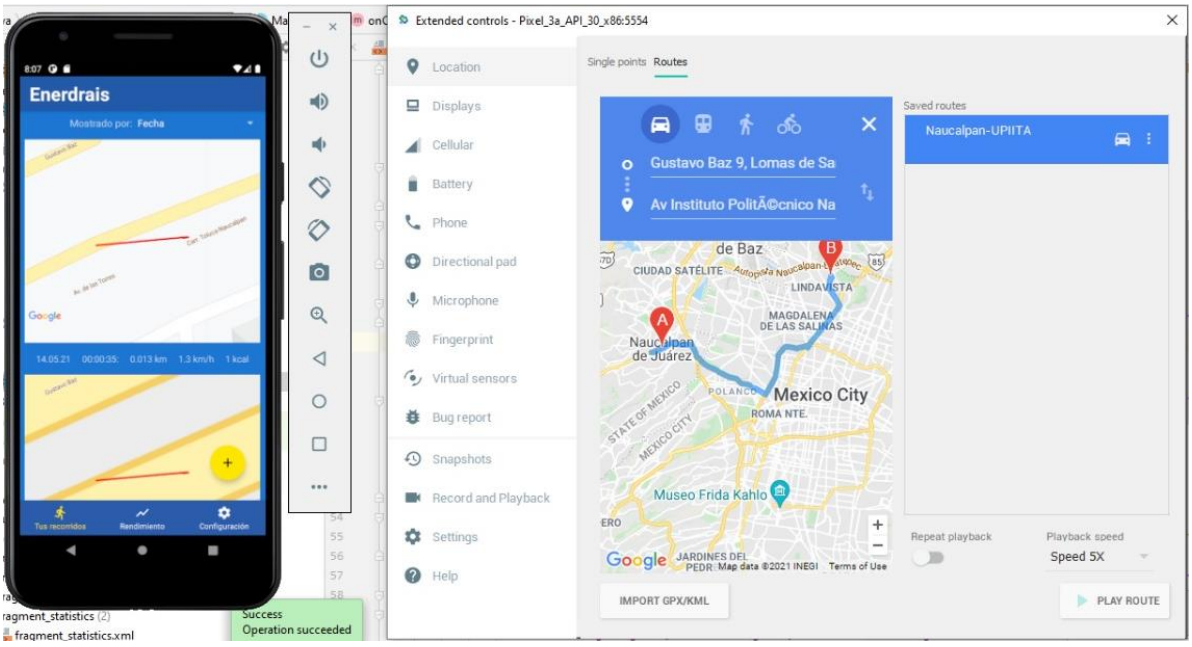


Figura 130. Captura de seguimiento de ruta de la aplicación 5.

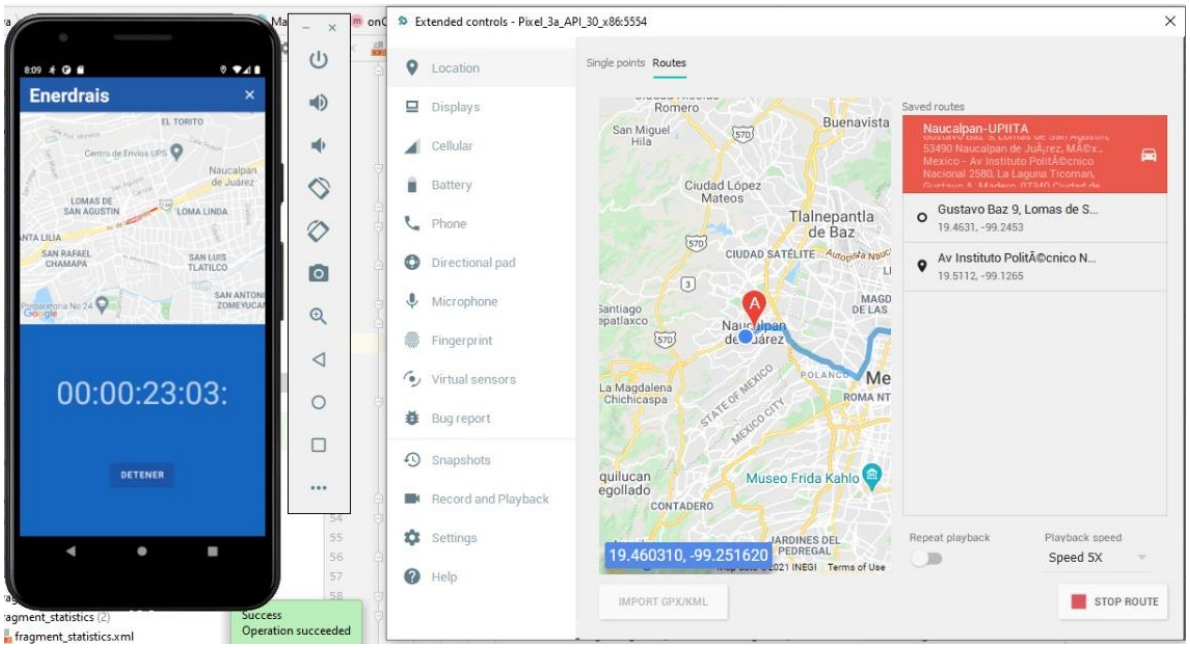


Figura 131. Captura de seguimiento de ruta de la aplicación 6.

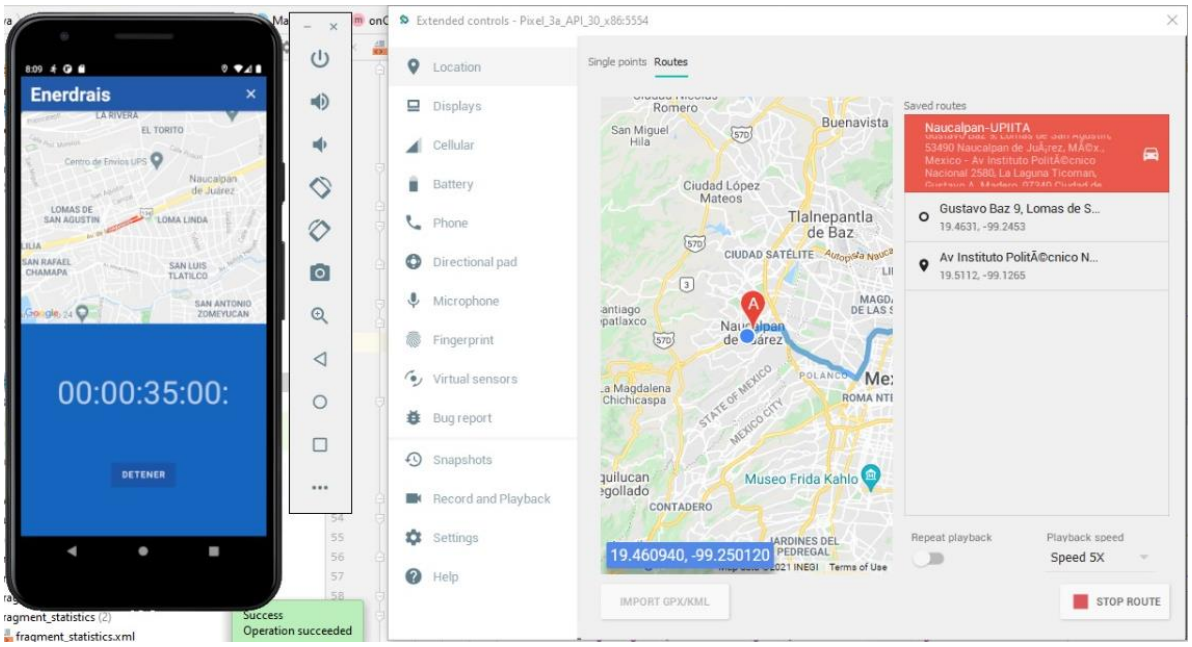


Figura 132. Captura de seguimiento de ruta de la aplicación 7.

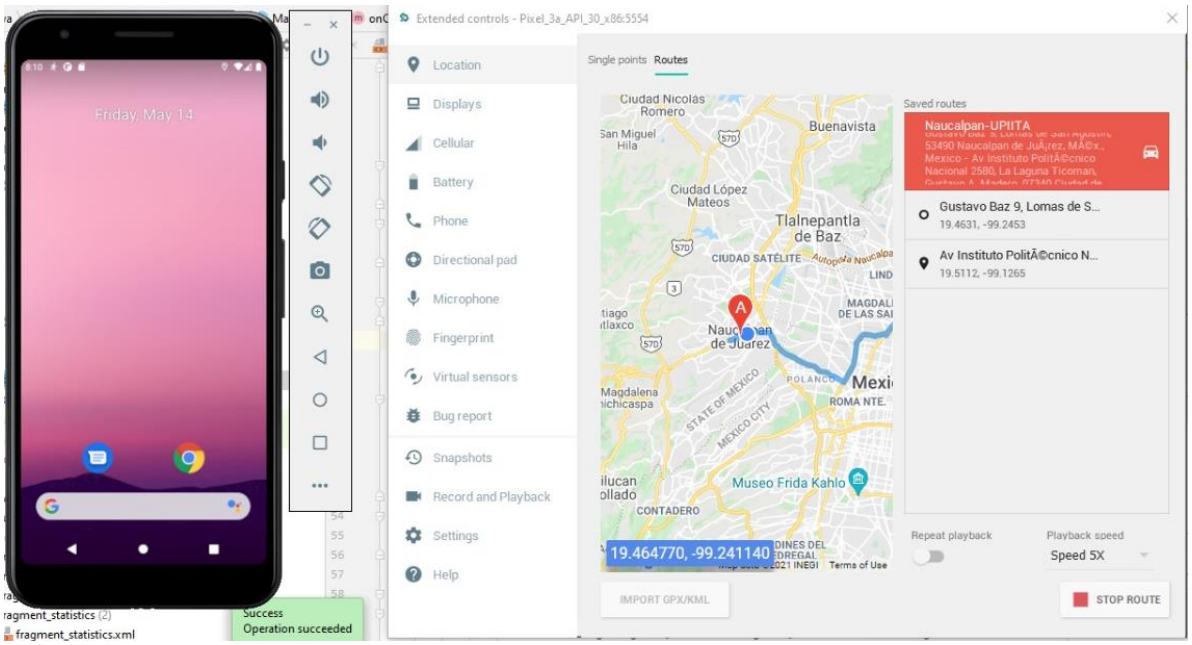


Figura 133. Captura de seguimiento de ruta de la aplicación 8.

Al cerrar la aplicación, el seguimiento de ruta se seguirá ejecutando en segundo plano y el tiempo transcurrido de la ruta aparecerá en forma de notificación, por lo que el usuario podrá bloquear su celular y la aplicación seguirá monitoreando su ruta, además por medio de la ventana de notificación se puede pausar o reanudar el monitoreo del recorrido sin necesidad de ingresar a la interfaz de usuario de la aplicación Android (figuras 134, 135 y 136).

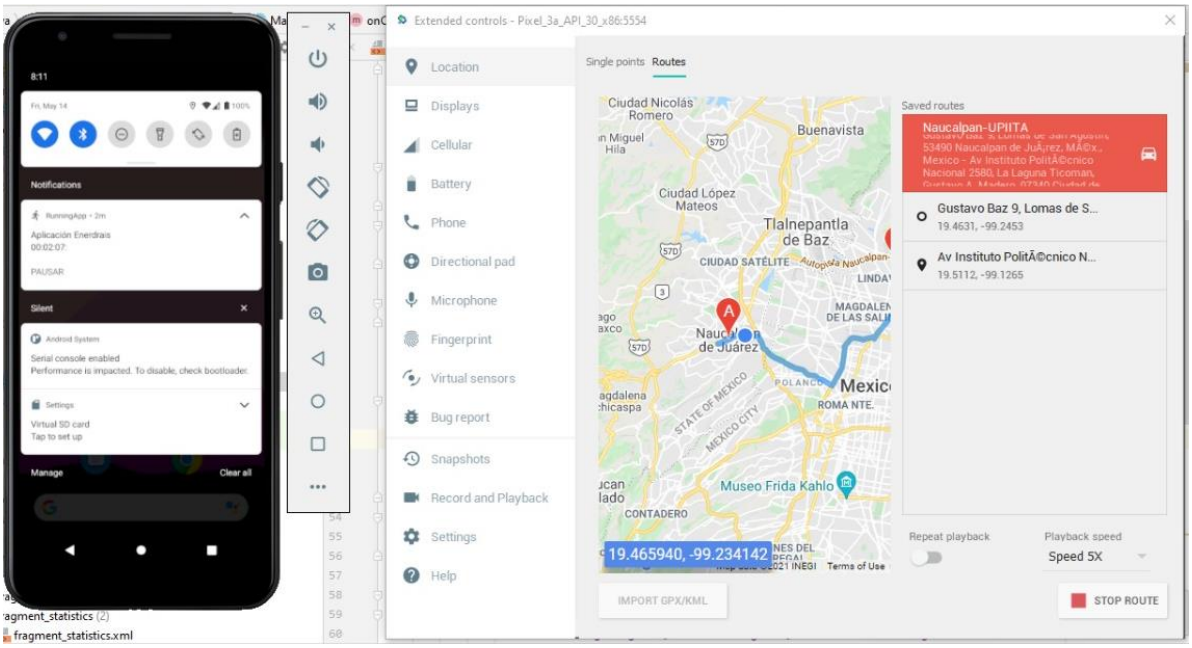


Figura 134. Captura de seguimiento de ruta de la aplicación 9.

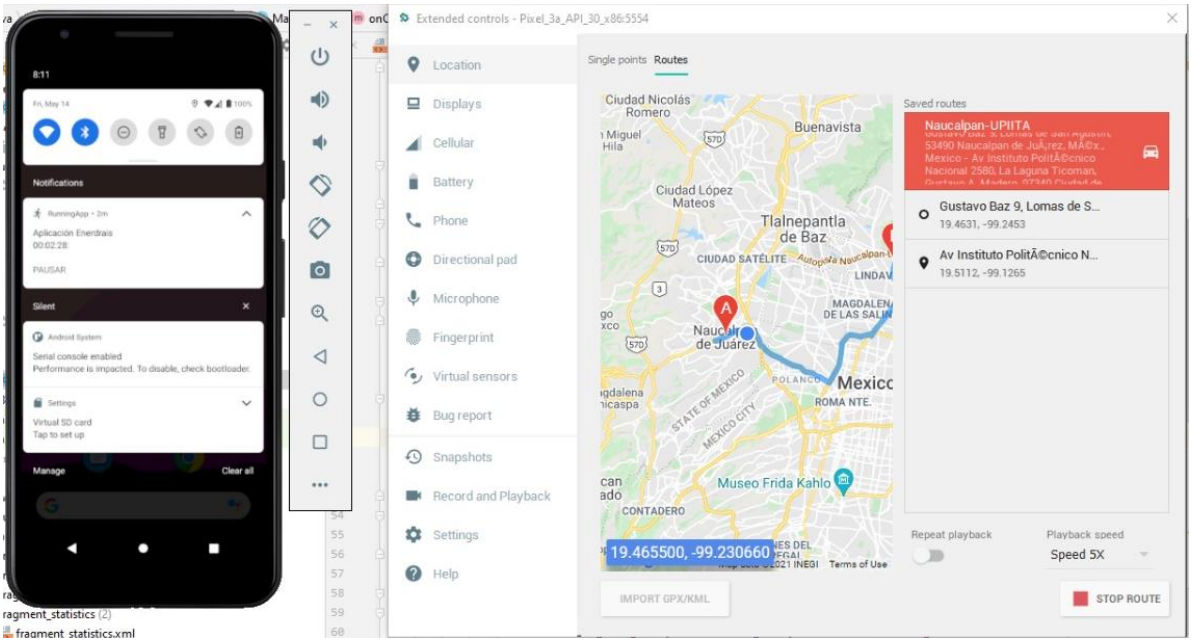


Figura 135. Captura de seguimiento de ruta de la aplicación 10.

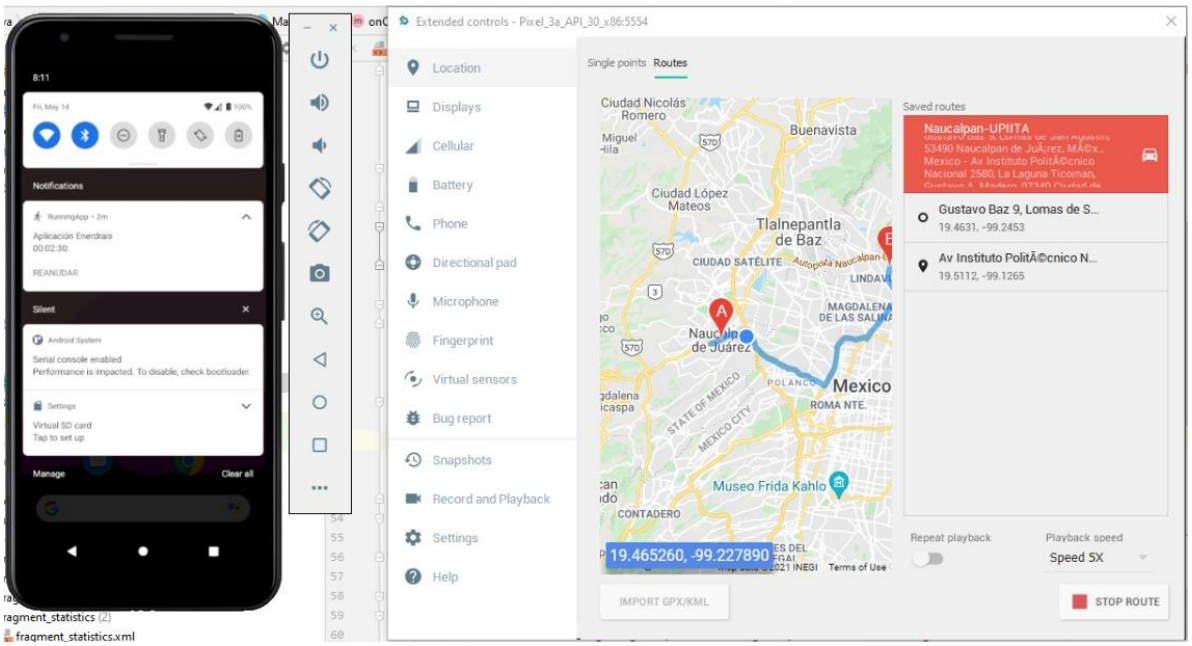


Figura 136. Captura de seguimiento de ruta de la aplicación 11.

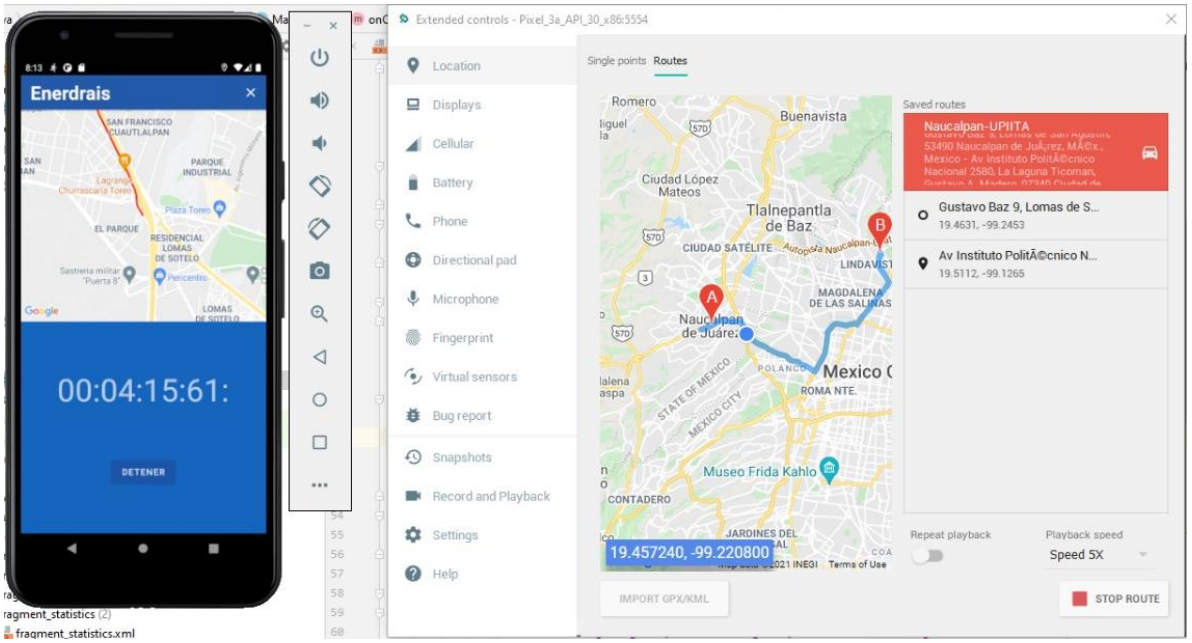


Figura 137. Captura de seguimiento de ruta de la aplicación 12.

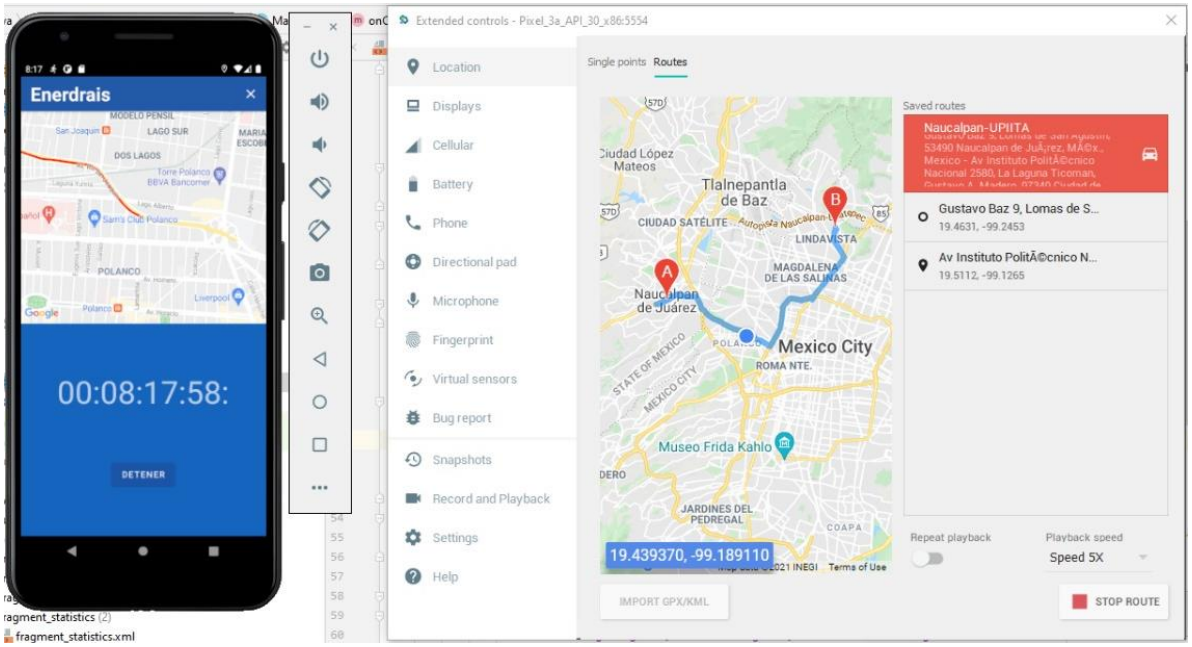


Figura 138. Captura de seguimiento de ruta de la aplicación 13.

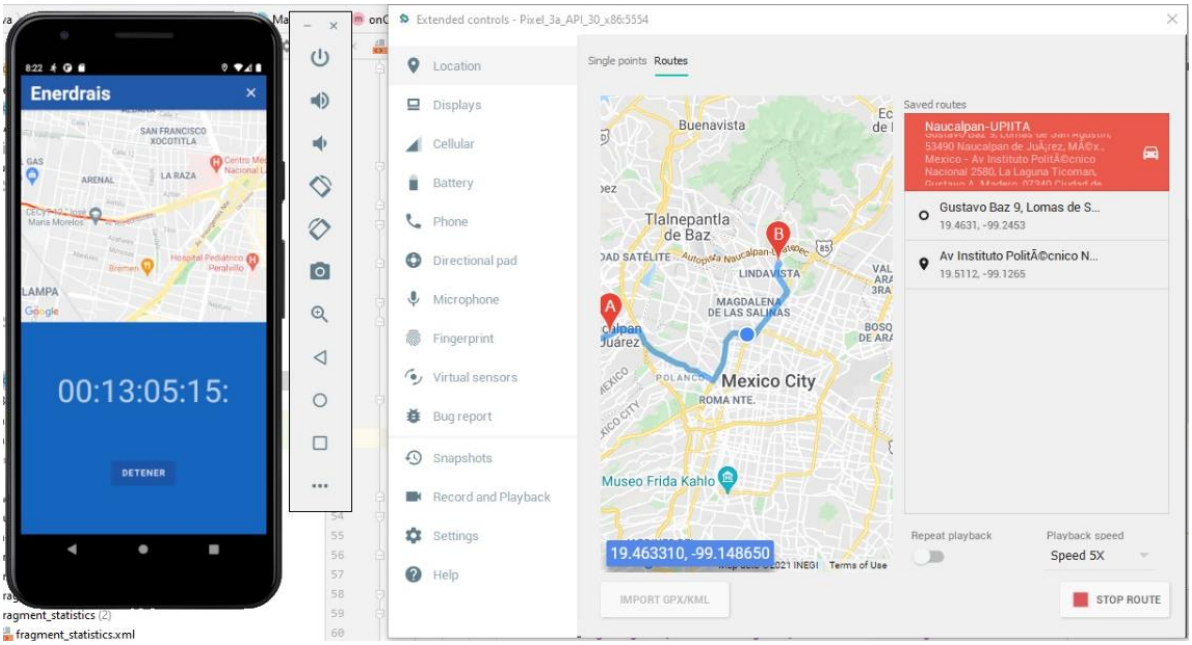


Figura 139. Captura de seguimiento de ruta de la aplicación 14.

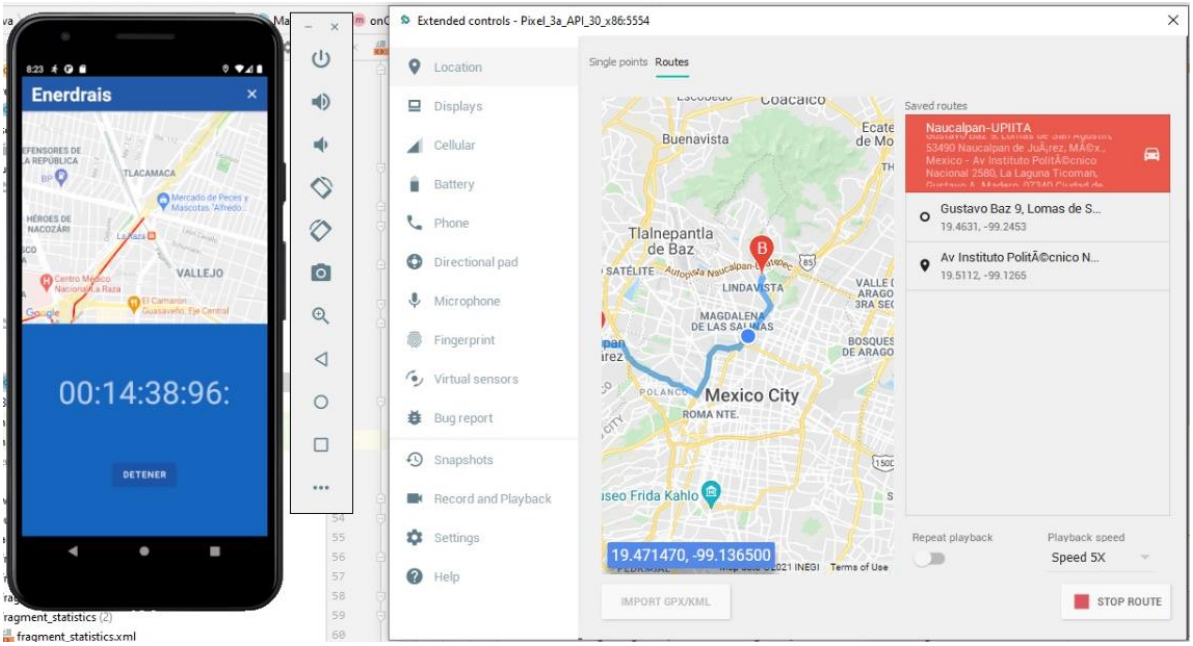


Figura 140. Captura de seguimiento de ruta de la aplicación 15.

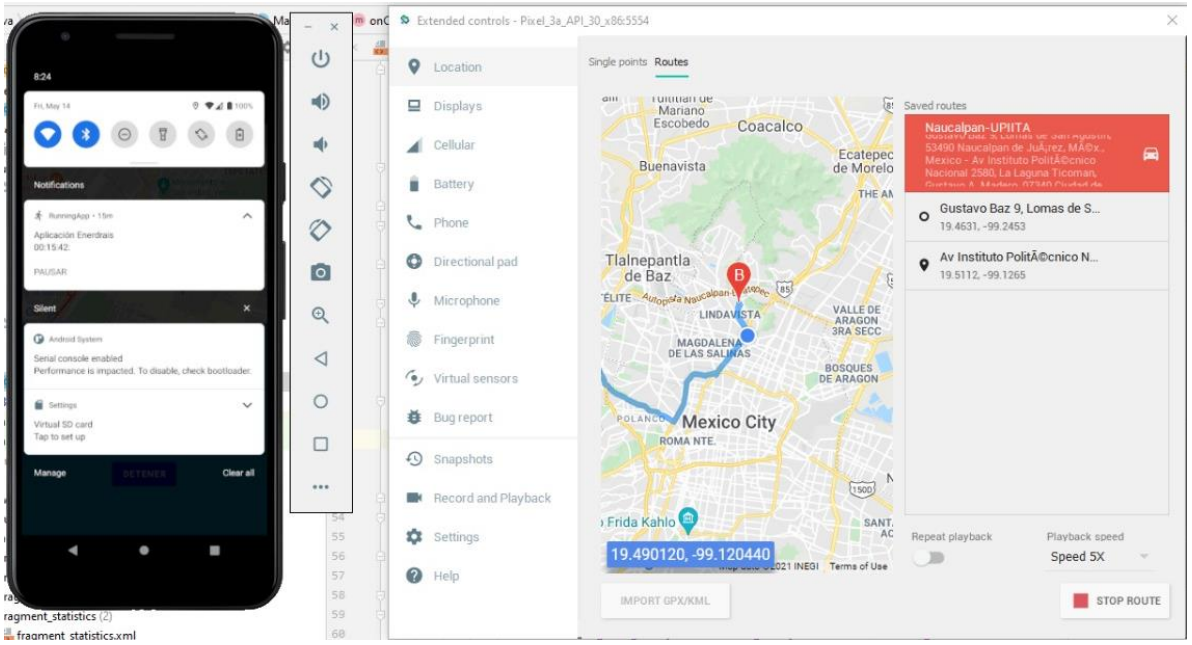


Figura 141. Captura de seguimiento de ruta de la aplicación 16.

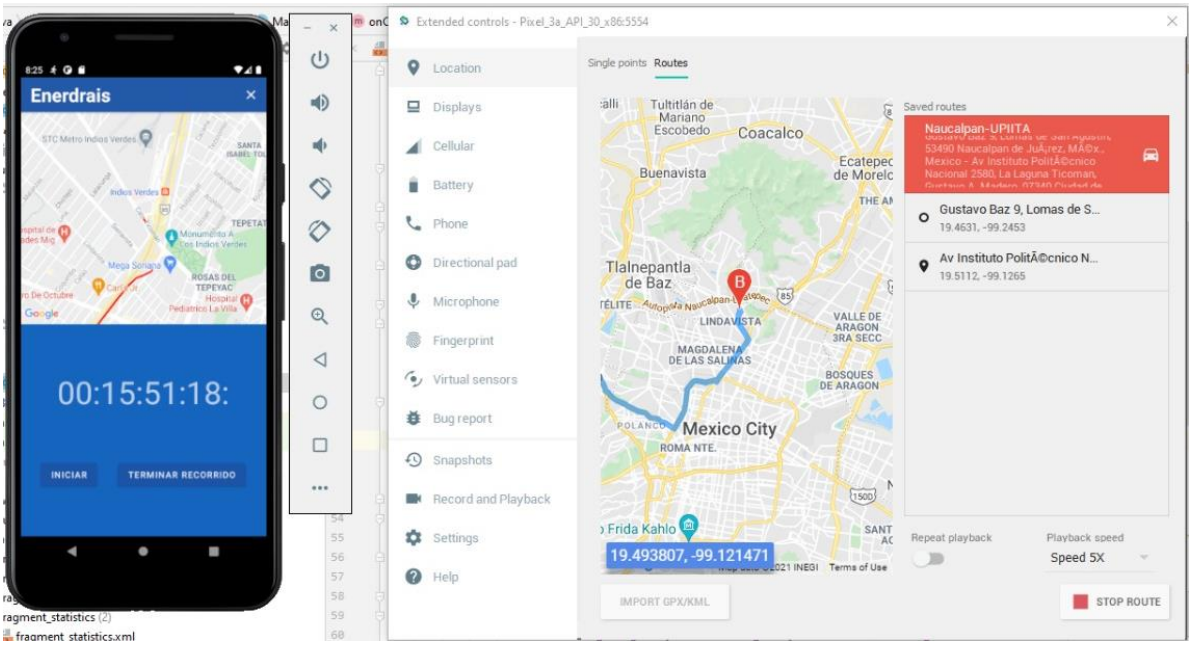


Figura 142. Captura de seguimiento de ruta de la aplicación 17.

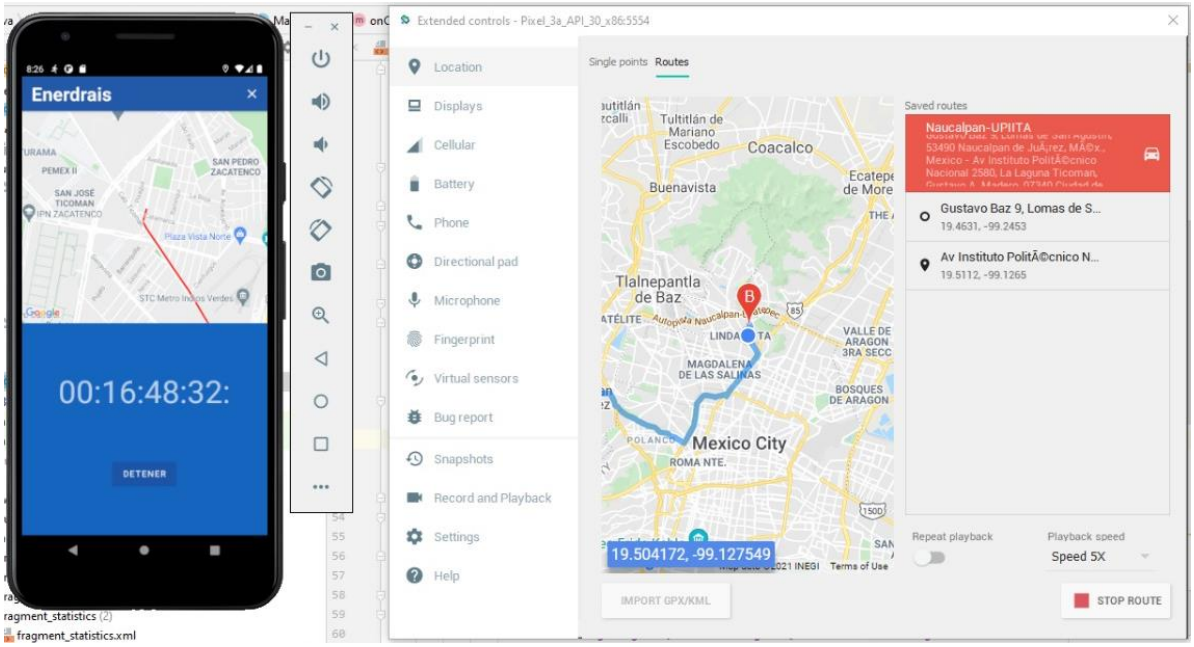


Figura 143. Captura de seguimiento de ruta de la aplicación 18.

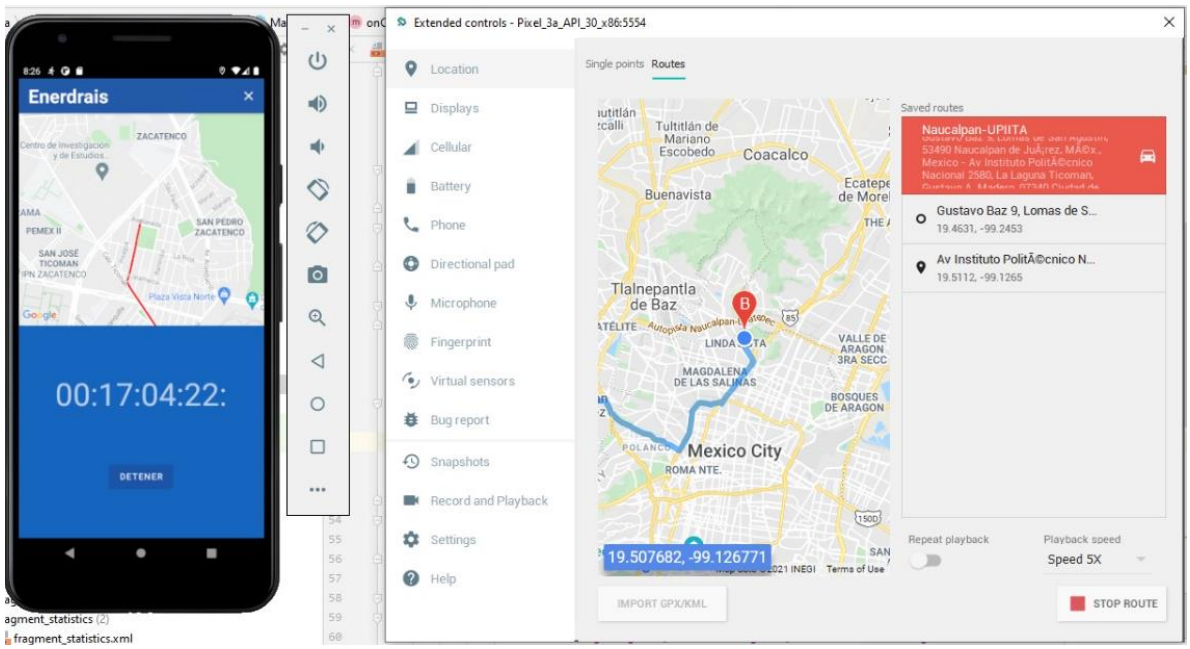


Figura 144. Captura de seguimiento de ruta de la aplicación 19.

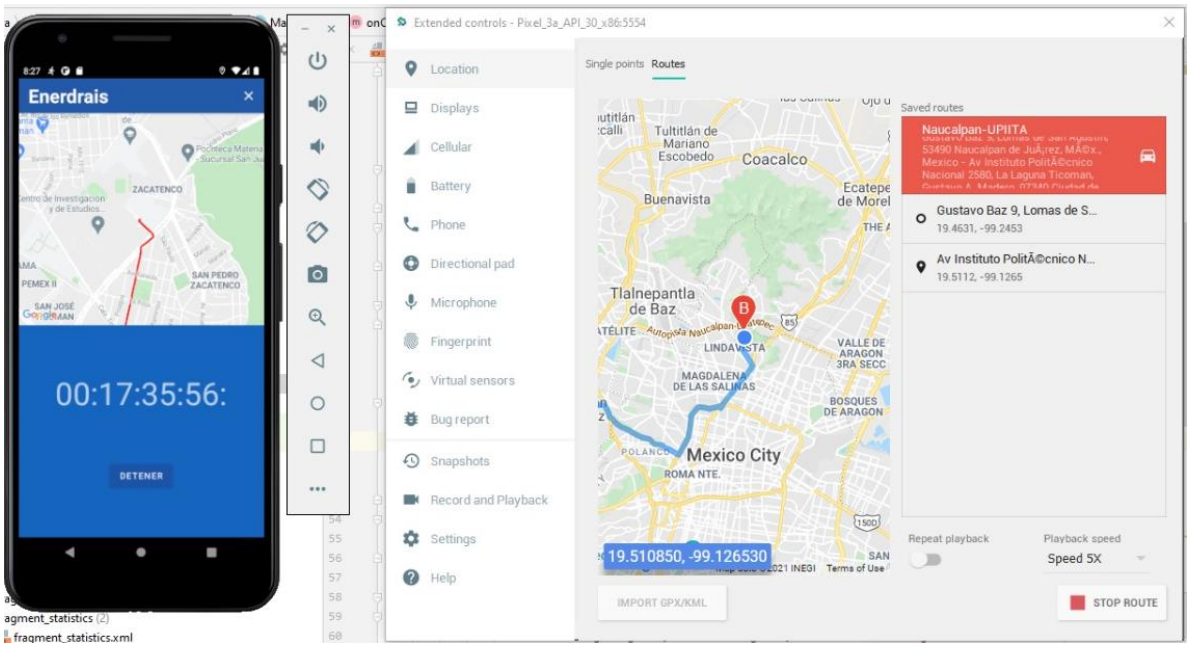


Figura 145. Captura de seguimiento de ruta de la aplicación 20.

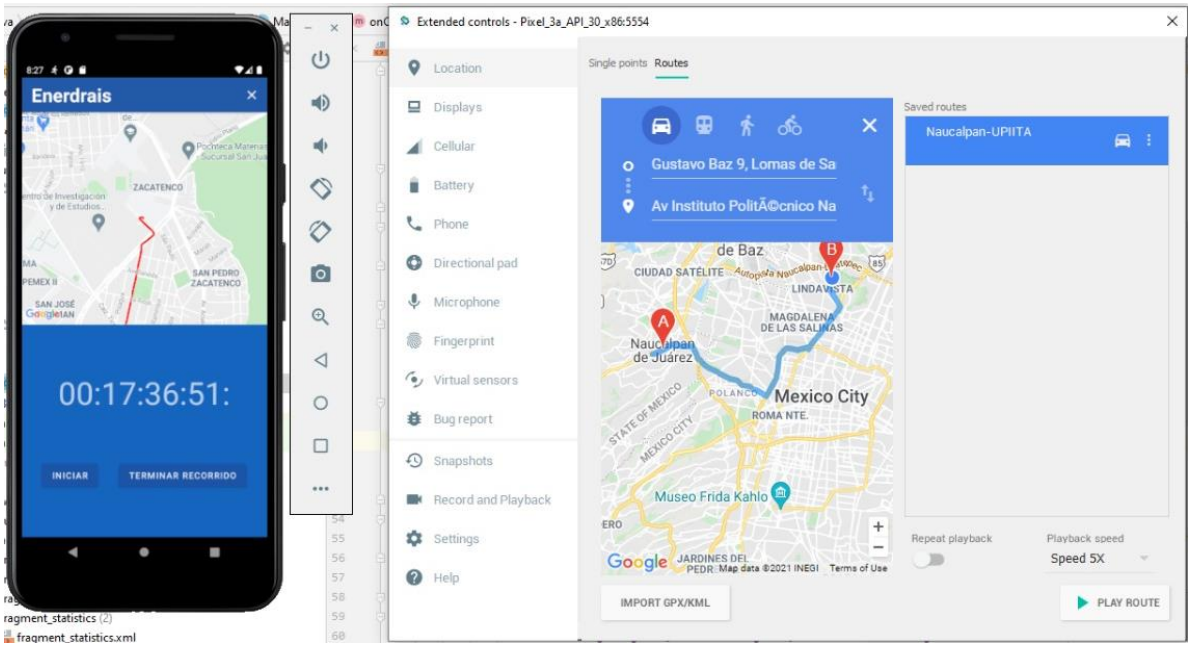


Figura 146. Captura de seguimiento de ruta de la aplicación 21.

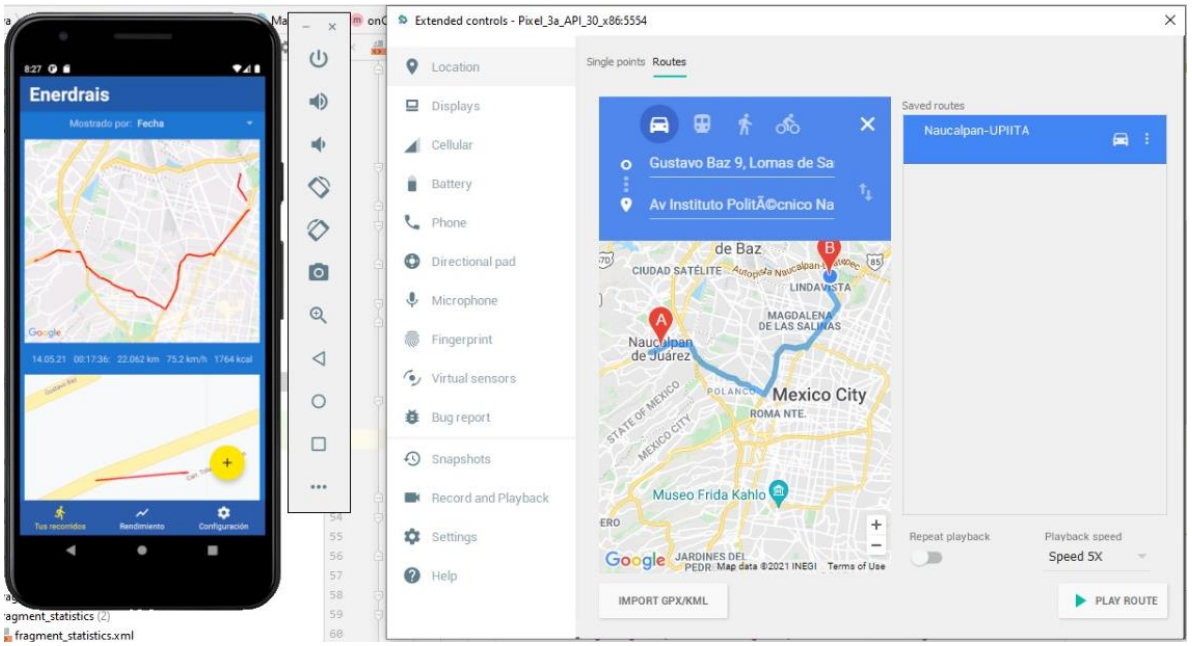


Figura 147. Captura de seguimiento de ruta de la aplicación 22 – Comparación de ruta entre el simulador del AVD y le ruta trazada por el mapa de la aplicación.

Al terminar un recorrido, este queda almacenado en la base de datos SQLite creada por medio de la librería Room, que almacena los datos dentro del dispositivo móvil en vez de hacerlo en un servidor remoto, como normalmente se hace con las bases de datos habituales. Se realiza el almacenamiento de datos para que el usuario pueda visualizar el historial de sus distintos recorridos y los pueda ver en función de la fecha cuando se realizó, la distancia recorrida, el tiempo de duración del recorrido, la velocidad promedio o las calorías quemadas. Se visualizará el recorrido junto con una captura de pantalla del mapa con la ruta dibujada (figura 145). Además, se podrá ver en la ventana de Rendimiento los datos de los distintos recorridos en el historial en forma de gráfica de barras (figuras 147 y 148).

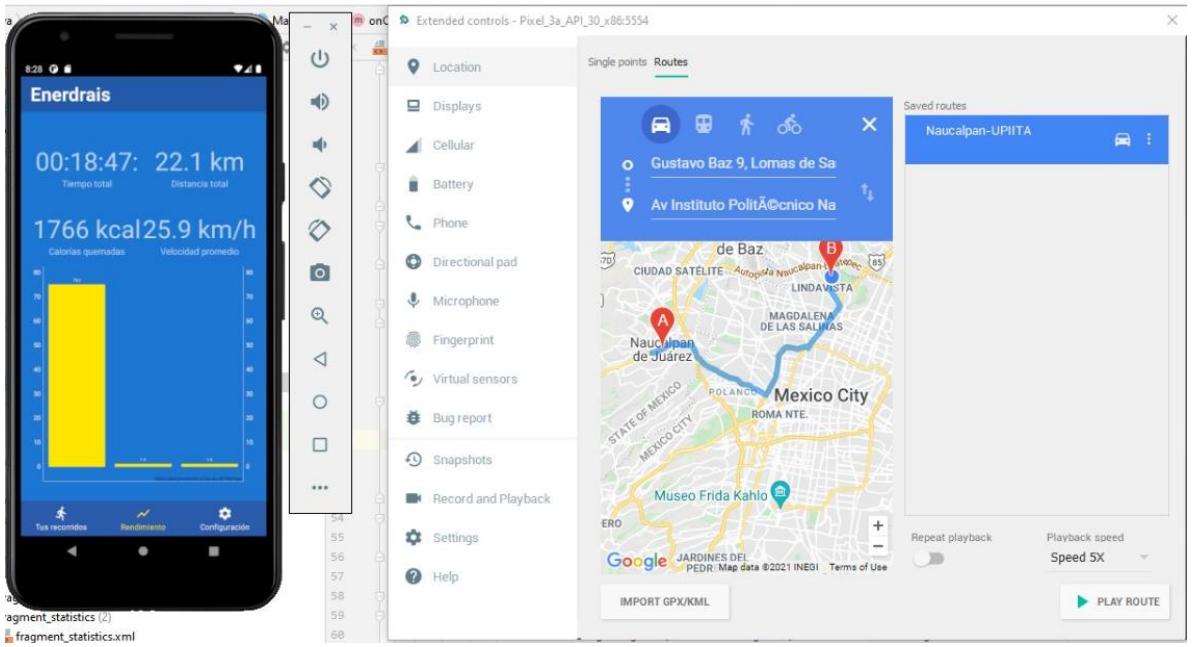


Figura 148. Captura de seguimiento de ruta de la aplicación 23.

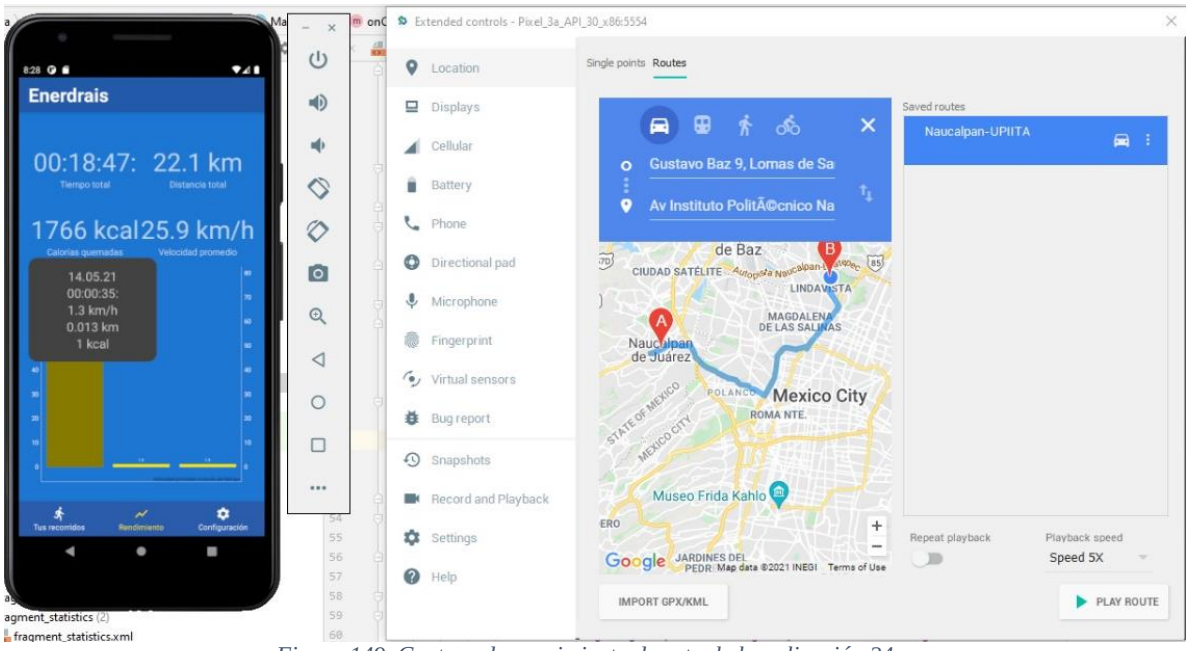


Figura 149. Captura de seguimiento de ruta de la aplicación 24.

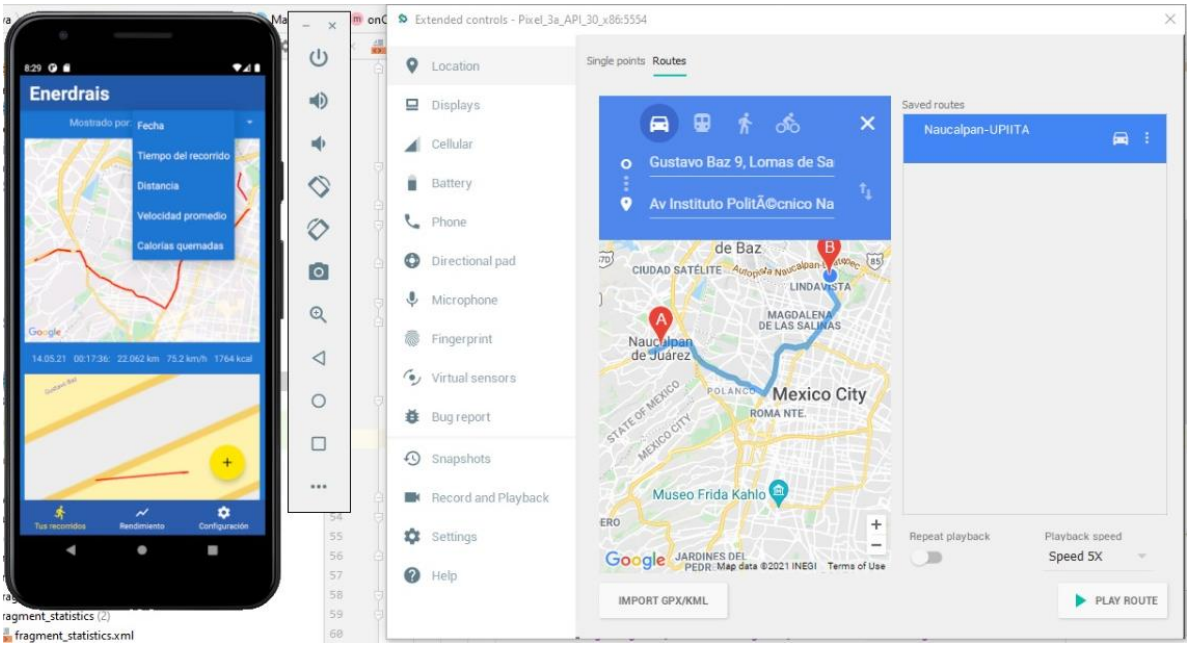


Figura 150. Captura de seguimiento de ruta de la aplicación 25.

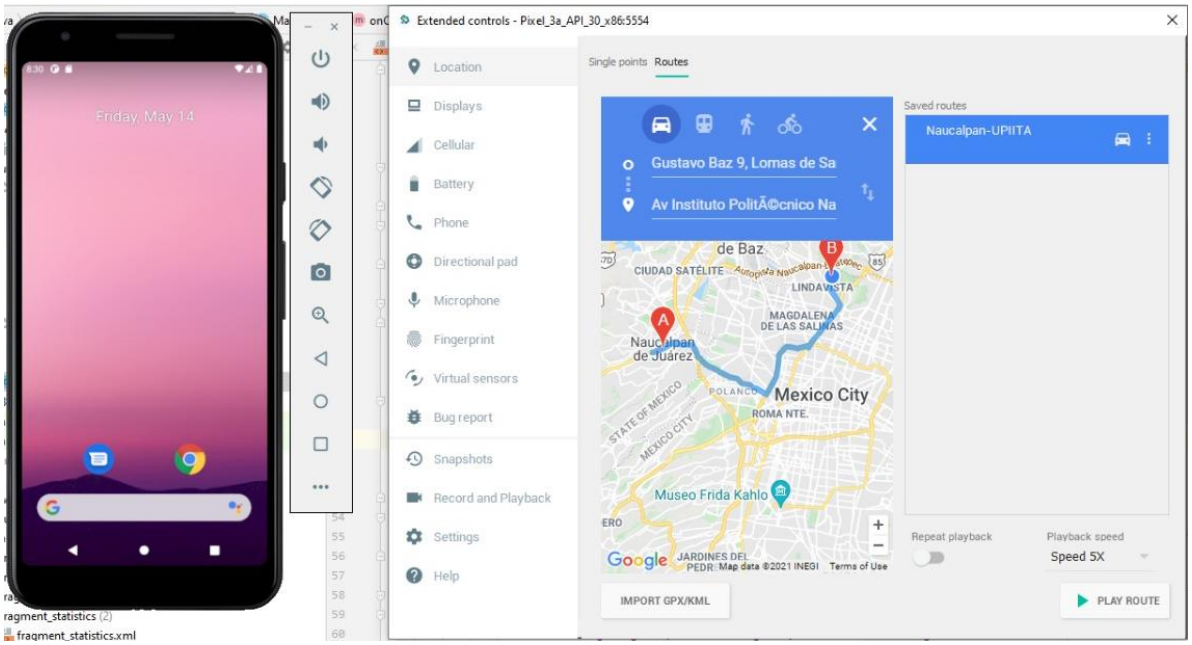


Figura 151. Captura de seguimiento de ruta de la aplicación 26.

a.3. Simulaciones del sistema de captación y acondicionamiento de energía eléctrica

A continuación, se presentan las simulaciones por separado de cada parte del circuito, ya que se están utilizando borneras para simular la conexión con los convertidores CC-CC elevador y reductor comerciales.

Simulación de valor del máximo a usar

El capacitor mínimo que se puede seleccionar es 56uF ya que en la simulación se muestra que la corriente pico – pico es de 1.48A, por lo tanto, es una corriente máxima, ya que en el diodo de rectificación solo puede circular como máximo 1A. Se seleccionará como máximo valor un capacitor de 56uF para evitar que los diodos se dañen (figura 152).

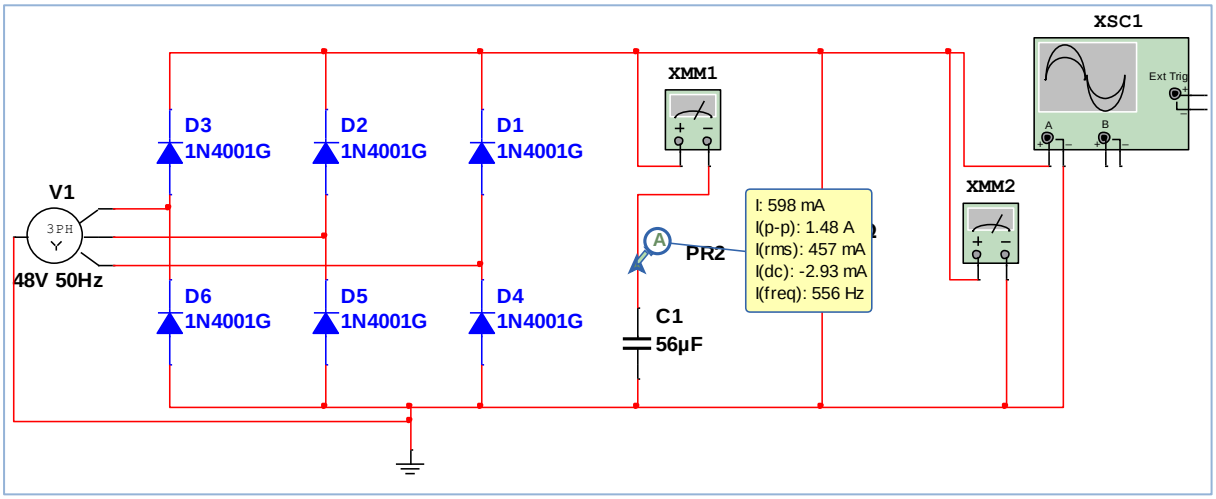


Figura 152. Simulación de puente de diodos para determinar el capacitor máximo para evitar un daño en los diodos.

Simulación del uso del relevador con microcontrolador

En esta simulación se tomaron en cuenta los valores calculados para determinar si el relevador se activará cuando circule la corriente calculada por su bobina. El funcionamiento es correcto, además se activará por defecto la acción de dirigir la señal hacia el convertidor reductor con el fin de alimentar la batería portátil. Por lo tanto, se detendrá el flujo de la señal hacia el convertidor reductor cuando no se active el relevador (figura 153).

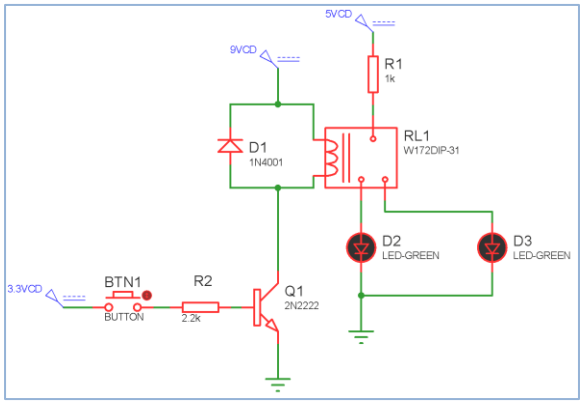


Figura 153. Prueba de funcionamiento del relevador con el microcontrolador ESP32.

En la siguiente figura se muestra la prueba de prueba de un generador XF07 con la parte de rectificación y el control de la ESP32 para controlar el relevador. Aquí se muestra que se generan 10V de tensión.

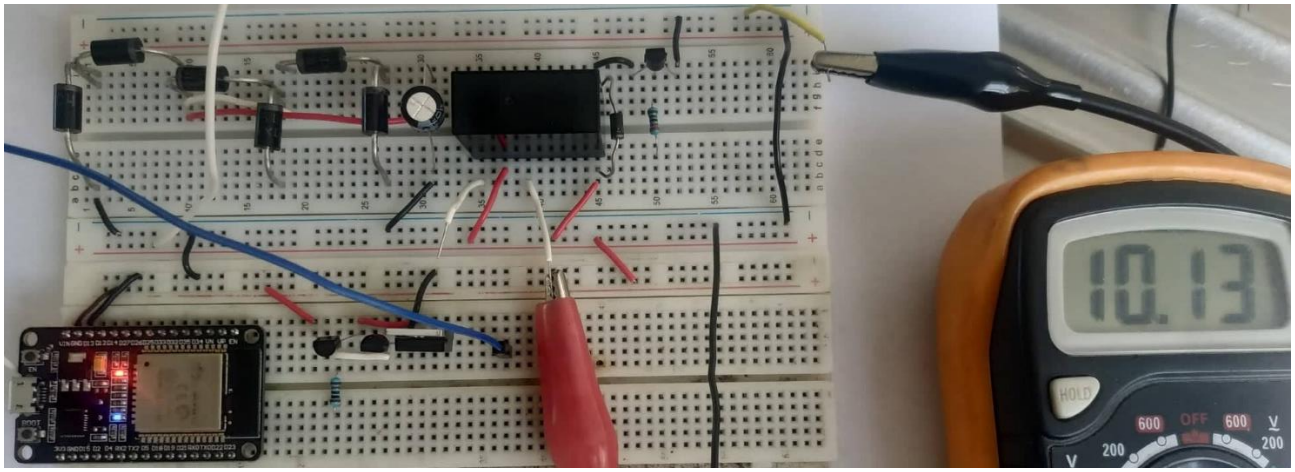


Figura 154. Prueba de relevador y restificación de señal.

Tensiones máximas de los convertidores en los divisores de tensión para habilitar la lectura que realiza el microcontrolador ESP32

La salida de tensión máxima del convertidor elevador es de 83V, por lo tanto, se comprobó la salida del divisor de tensión, para determinar el valor de la resistencia R2 y así obtener una tensión máxima de 3.3V cuando a la entrada de R1 se aplican 83V (figura 155).

Lo mismo se simulo para el convertidor reductor, el cual puede dar una tensión de 37V, por lo tanto, se simulo con tensión máxima de 37V, a la salida del divisor de tensión se puede observar que la tensión máxima es de 3.3V para que no dañe al microcontrolador ESP32 (figura 156).

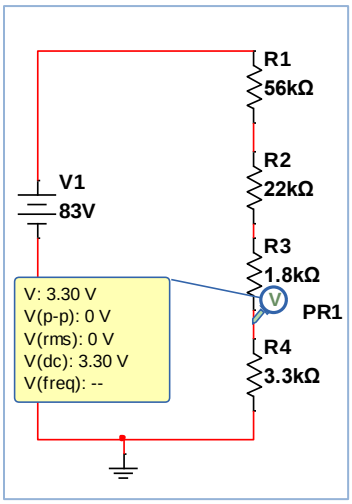


Figura 155. Divisor de tensión para un convertidor elevador.

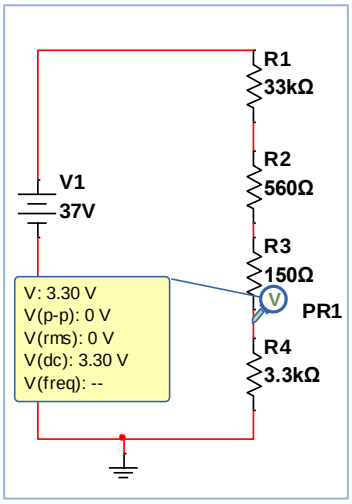


Figura 156. Divisor de tensión para tensiones de salida máxima de un convertidor reductor.

Prueba de control de Mosfet como compuertas

En la siguiente figura se mustra el control y prueba del mosfet como compuerta de paso de la tensión de alimentación a la batería de bicicleta. Como se puede ver a la salida se tiene una tensión de 49.9V de tensión por lo cual recomendable para alimantar la batería de bicicleta. Se controlo el mosfet con ayuda de transistor TIP41. Se utilizo de la manera que al prender la maquina simpre se mantega cerradas las compuertar y que cuando miconcontrolador se le mande la señal deje alimantar la batería.

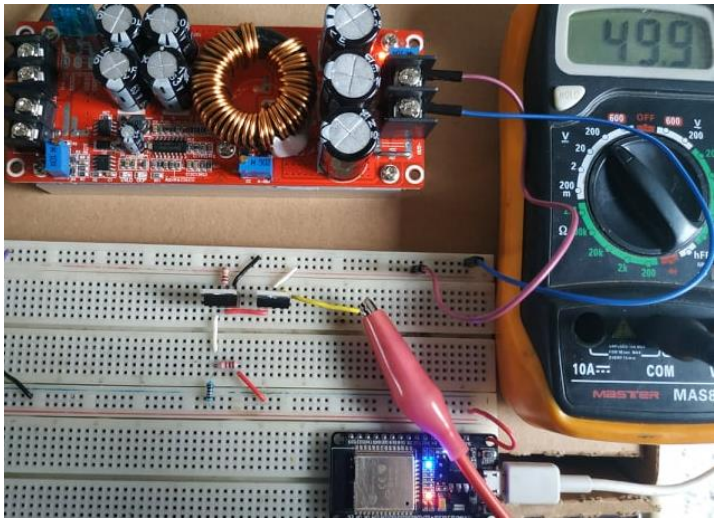


Figura 157. Control de salida de convertidor elevador con ESP32.

En la siguiente figura podemos observar el control de alimentación para la batería portátil o otros dispositivos que consumen 5V de tensión. Se realiza el control por parte del usuario al mandarle una

orden al microcontrolador ESP32, el microcontrolador manda la señal al transistor 2N2222 que activa el mosfet par abrir la conmpuerta y alimentar algun dispositivo.

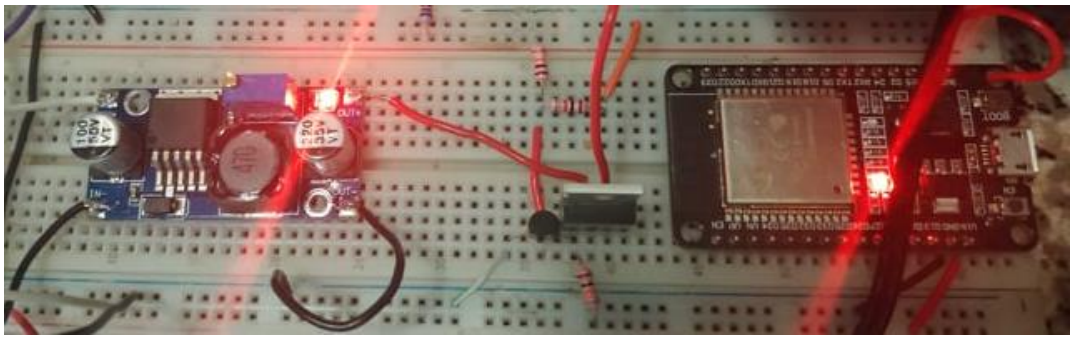


Figura 158. Control de convertidor reductor con ESP32.

Simulacion de la programación del microcontrolador EPS32

Cuando el microcontrolador inicia, empieza a parpadear un led interno que se encuentra en el pin2, el cual indica que se está esperando a que el usuario se conecte con su dispositivo móvil mediante una conexión Bluetooth (figura 159).

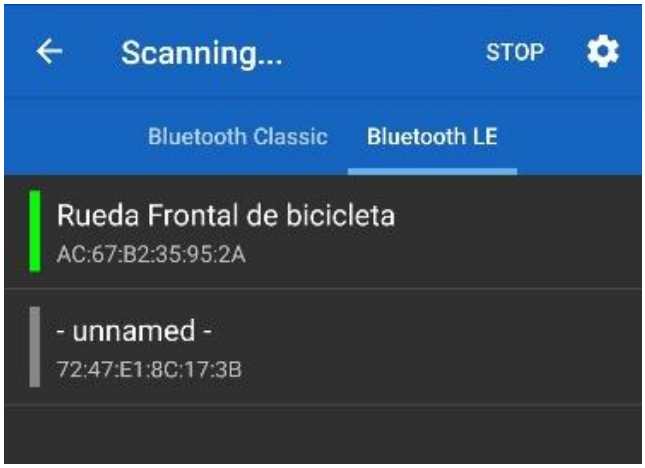


Figura 159. Escaneo bluetooth para conexión de rueda generadora y dispositivo inteligente.

Se utilizó la aplicación “Serial Bluetooth Terminal” para comprobar el funcionamiento en un dispositivo móvil con sistema operativo Android, para ello primero se realiza el escaneo como se muestra en la figura 144. En este caso el nombre del dispositivo Bluetooth programado en el microcontrolador ESP32 fue llamado como “Rueda Frontal de bicicleta”.

Cuando se establece una conexión exitosa entre los dos dispositivos, el micrcontrolador mantendrá prendido el led interno del pin numero 2 (figura 160 y 161).

Posteriormente ya se puede realizar el envio de comandos, el primer comando es “power”, el cual activa la alimentación para la batería portátil y además de eso, realiza la lectura de la carga de la batería de litio, indicando si la tensión entregada es adecuada para la alimentar la batería portátil.

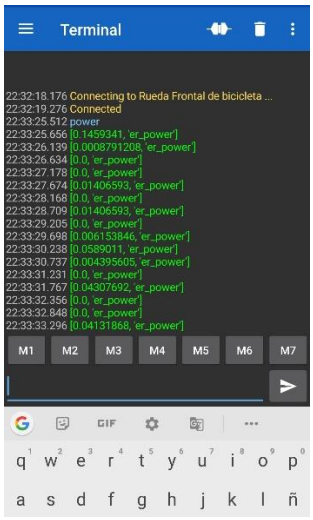


Figura 160. Simulación de envío, comando "power" tipo String.

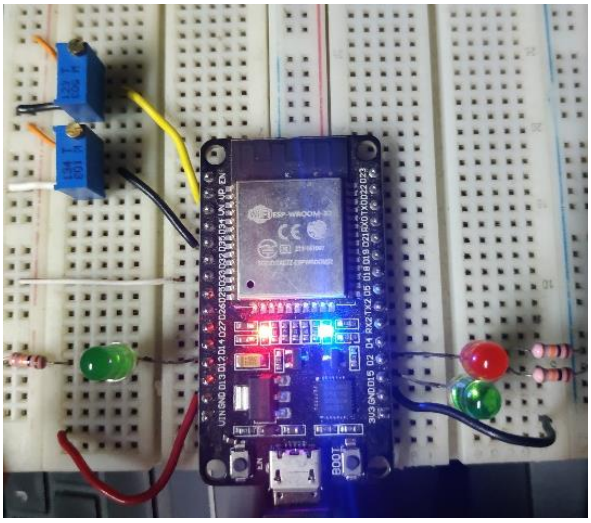


Figura 161. Simulación con componentes básicos.

Otro comando que existe es el de parar la alimentación hacia las dos baterias, el cual es “stop”, finalmente, el comando “bici”, hace que el circuito alimente la batería de la bicicleta eléctrica, por lo cual solo se envían los datos de tensión que tiene a la salida el convertidor elevador. El microcontrolador asegura que la tensión enviada a la batería de la bicicleta eléctrica sea correcta para evitar dañarla (figura 162).

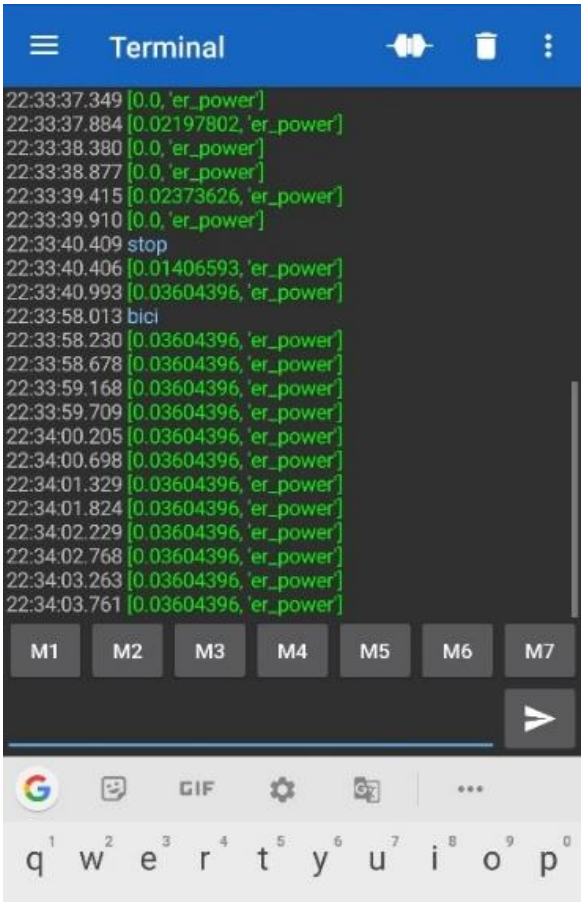


Figura 162. Simulación de comando "stop" y bici.

b. Análisis de costos

A continuación, se presenta la tabla 49 con el análisis de costos de los materiales para llevar a cabo la construcción de la rueda frontal de bicicleta generadora de energía eléctrica.

Tabla 50. Análisis de costos.

Análisis de costos	
Descripción	**Monto aproximado**
Bicicleta Monk Kron R26 18V	$2,879
Buje de bicicleta con eje de cierre rápido	$458.17
Filamento pla - fibra de carbono	$144
Espátulas para desmontar llantas de bicicleta	$115
Rines rodada 26 marca Kylin	$899
Bomba de pedal para inflar llantas de bicicleta	$169
3 transistores PN2222A	$11
2 transistores PN2907A	$30
3 capacitores electrolíticos 100 µF @50V	$9
4 diodos 1N4001	$4
2 inductores 100uH @ 10A	$24
2 resistencias de 47 Ω	$4
2 IRF9540	$39
2 IRF540	$39
Correa trapezoidal C64	$442
2 motores MXUS-XF07	$5,100
Lámina de aluminio 3003 H14 - cal 10: 0.91 X 2.44 m	$2,751.00
Lm2596 módulo regulador de tensión CC-CC Buck	$98
Módulo regulador de tensión CC-CC Boost de 1200W	$200
Barra circular de Aluminio 6061-T6 - tramo de 25 cm.	$169
Costo total, tentativo, (usando lámina de aluminio 3003 H14 cal 14)	$13,685.67

c. Impacto ambiental

El impacto social del proyecto recae en que la energía gastada por miles de mexicanos al día pueda ser utilizada por ellos mismos, cargando sus dispositivos móviles aun cuando se encuentren fuera de casa,

impulsando la actividad física en la sedentaria vida de los citadinos, proporcionándoles un beneficio extra al hacerlo y donde durante la ruta a sus trabajos o que haceres puedan aportar al medio ambiente con una generación y utilización de energía limpia.

d. Sustentabilidad

Según el Instituto Nacional de Estadística y Geografía (INEGI) en 1950, poco menos del 43% de la población de México vivía en localidades urbanas, en 1990 era el 71% y para 2010, esta cifra aumentó a casi el 78%. Lo anterior ha provocado una congestión de tráfico, no solo en las principales ciudades de México, sino en gran parte del mundo y todo indica que seguirá agravándose, constituyendo un peligro para la calidad de vida de las personas. El transporte alternativo como bicicletas, patines y motonetas eléctricas se ha convertido en una opción adicional a los tradicionales vehículos automotor (Metro, autobús y automóvil) con el fin de moverse más rápidamente, reducir la contaminación y hacer más sustentable la calidad de vida [38], la rueda frontal generadora de energía incentiva al uso de este tipo de transporte ya sea mecánico o eléctrico ya que si se está utilizando una bicicleta mecánica el usuario podrá cargar su dispositivo móvil, monitorear su velocidad, ruta y calorías quemadas, además de tener un registro en la aplicación asociada al proyecto donde pueda comparar su rendimiento en los distintos recorridos que haya realizado y si está utilizando una bicicleta eléctrica podrá extender su tiempo de uso, logrando que pueda trasladarse grandes distancias sin la necesidad de subirse a un automóvil o motocicleta, motivándolo a seguir ese estilo de vida saludable y amable para el medio ambiente, reduciendo así la cogestión de tráfico en localidades urbanas y asegurando la calidad de vida de generaciones futuras.

CONCLUSIONES

Manufacturar con aluminio tiene como ventaja que es de los metales más ligeros en el mercado, su desventaja es que el precio es más elevado en comparación con otros metales como el acero y además su gran desventaja es que no es nada sencillo soldar aluminio, ya que por su bajo punto de fusión es muy fácil que el material se dañe al tratar de soldar, pero al desarrollar este proyecto, los autores del presente trabajo nos percatado que una gran opción para trabajar con aluminio es el uso de remaches, el problema es que hay que cuidar mucho los concentradores de esfuerzo que se pueden crear debido a los barrenos que se deben hacer para poder unir las piezas de esta forma, pero si el cálculo y la simulación de esfuerzos está hecho de una forma correcta, hacer ensamble de piezas de aluminio haciendo uso de remaches es una opción muy viable, logrando así que una estructura sea ligera y resistente.

El crear una aplicación móvil desarrollada con Android que utilice librerías o APIs es una tarea que requiere conocimiento acerca de las actualizaciones del editor de código Android Studio, ya que si no se domina estos temas, aunque el código esté bien pensado y sea funcional, se puede caer en un problema de dependencias, en donde simplemente por tener una versión distinta de graddle, todo el código puede fallar por un tema de actualizaciones, que fue uno de los problemas iniciales que se tuvo al querer desarrollar la aplicación móvil para la rueda frontal generadora de energía eléctrica usando las librerías de Dagger y Dagger Hilt.

Crear piezas mecánicas por medio de la impresión 3D es útil en cuestión de diseño, ya que no existe límite y se puede ahondar en el más mínimo detalle para dejar la pieza con un aspecto estético y funcional, el problema reside en que su comportamiento mecánico no está estandarizado, ya que la carga

que puede soportar una pieza creada con impresión 3D está muy ligada a la densidad del relleno de la impresión, la marca de la impresora, el filamento y el slicer que se utiliza para crear el gcode, por lo que es mejor para las piezas que deben soportar la mayor carga en una estructura usar metales, ya que sus propiedades mecánicas están muy estandarizadas y estudiadas, donde además se cuenta con manuales técnicos desarrollados por múltiples empresas en los que se exponen las propiedades de cada metal de su catálogo, pero para piezas cuya función es estar en fricción, como en este caso lo son las ruedas tipo polea laterales o el acoplamiento mecánico entre la banda de transmisión trapezoidal C64 y el generador eléctrico MXUS-XF07, es de gran utilidad manufacturar las piezas con impresión 3D ya que se le puede dar cierta rugosidad a su superficie para ayudar en la fricción, hacerla de cualquier tamaño y si se le da un relleno del 80% se puede aumentar la resistencia de la pieza.

Debido a la baja eficiencia obtenida en la señal obtenida cuando se diseña desde cero un convertidor CC-CC reductor o elevador controlado por medio de una señal PWM ya que se necesita utilizar elementos diseñados específicamente para el circuito, se tomó la decisión de utilizar convertidores elevadores y reductores que se encuentran en el mercado, debido a que los elementos que conforman la placa de la fuente conmutada son únicos y están diseñados específicamente para esa función, como por ejemplo el inductor, que debe ser especialmente diseñado para satisfacer las necesidades de corriente que debe cumplir el convertidor, por eso es que los convertidores reductores y elevadores encontrados se cuenta con un inductor toroidal, ya que está diseñado específicamente para ser utilizado en una fuente conmutada.

Finalmente, el diseño del proyecto electromecánico es factible, las piezas mecánicas son ligeras y se pueden manufacturar por medio de un CNC de plasma, por lo que su producción en masa es posible y con gran calidad en el producto final, las simulaciones del circuito indican que puede entregar la tensión necesaria para el la batería portable o la batería de la bicicleta eléctrica y la aplicación realiza correctamente el seguimiento de ruta, monitoreo de tiempo transcurrido, distancia y velocidad promedio durante el recorrido.

REFERENCIAS

[1] Soonerbike, *"Introducción a la bicicleta eléctrica",* Soonerbike, 2018 [Online]. Available: http://www.soonerbike.com/introduccion.html. [Último acceso: 23 noviembre 2020].

[2] Página oficial de Patinetes eléctricos y movilidad, *"Infografía: Historia de las bicicletas* eléctricas", 2020 [Online]. Available: https://myscoot.eu/blog/historia-de-las-bicicletas-electricas. [Último acceso: 23 noviembre 2020].

[3] Forbes México, A. Solís, 2018. *Estas Empresas Pelean Por La Movilidad De La CDMX En Cada Pedaleada* [Online]. Available: https://www.forbes.com.mx/estas-firmas-pelean-la-movilidad-en-cada-pedaleada/

[4] Procuraduría Federal del Consumidor, *Transporte alternativo. Moverse de manera diferente*, 2019. [Último acceso: 29 octubre 2020].

[5] Página oficial de ECOBIBI, 2020 [Online]. Available: https://www.ecobici.cdmx.gob.mx. [Último acceso: 18 noviembre 2020].

[6] Superpedestrian support center, Copenhagen Wheel Charging, 2018. [Online]. Available: https://account.superpedestrian.com/support/article/Charging-Partners#:~:text=From%20empty%2C%20the%20Wheel%20will,supports%20the%20Copenhagen%20Wheel's%20battery. [Último acceso: 26 diciembre 2020].

[7] Solís Lara, P., 2017. *Sistema Híbrido Electromecánico Para Transporte Y Generación De Energía*. Licenciatura. IPN - ESIME. [Último acceso: 29 octubre 2020].

[8] Mata García, D. and Barrera Figueroa, J., 2018. *Implementación De Un Sistema De Carga Y Almacenamiento De Energía Eléctrica Aprovechando La Energía Cinética De Una Bicicleta*. Licenciatura. IPN - ESIME Zacatenco. [Último acceso: 29 octubre 2020].

[9] Rueda Alcalá, J., 2017. *Análisis Para La Recuperación De Energía No Aprovechada Por Medio Del Rin De Una Bicicleta*. Licenciatura. IPN - ESIME Azcapotzalco. [Último acceso: 29 octubre 2020].

[10] Bernal Méndez, P. and Torres Alvarracín, J., 2019. *Diseño E Implementación De Un Sistema De Asistencia Eléctrico En Una Bicicleta Mediante La Reutilización De Baterías De Ni-MH*. Licenciatura. Universidad Politécnica Salesiana. [Último acceso: 29 octubre 2020].

[11] R. Ballantine and R. Grant, *El gran libro de la bicicleta*. Madrid: El País, 1992, p. 154. [Último acceso: 29 octubre 2020].

[12] Sebastián, V., 2019. *Esta Rueda Convierte En Bicicleta Eléctrica Cualquier Bici Tradicional Siempre Que Estés Dispuesto A Pagar 2000 Euros*. [online] Xataka.com. Available at: <https://www.xataka.com/vehiculos/esta-rueda-convierte-bicicleta-electrica-cualquier-bici-tradicional-estes-dispuesto-a-pagar-2000-euros> [Accessed 20 January 2021].

[13] Ovrik. 2018. *Urbanx, Para Convertir Cualquier Bicicleta En Una Eléctrica - Ovrik*. [online] Available at: <http://www.ovrik.com/2017/03/20/urbanx-para-convertir-cualquier-bicicleta-en-una-electrica/> [Accessed 20 January 2021].

[14] V. SÁNCHEZ, "Partes de una Bicicleta y sus Funciones que todo ciclista debe saber", *Terránea*, 2020. [Online]. Available: https://blog.terranea.es/partes-bicicleta/. [Último acceso: 25 octubre 2020].

[15] "Rueda de bicicleta", *Es.wikipedia.org*, 2020. [Online]. Available: https://es.wikipedia.org/wiki/Rueda_de_bicicleta#cite_ref-7. [Último acceso: 25 octubre 2020].

[16] Internet de las cosas, Moisés Barrio Andrés, 2018, Universidad Carlos III de Madrid. [Último acceso: 24 noviembre 2020].

[17] S. Chapman, C. Rodríguez Pérez and Santana Díaz Alfredo., *Máquinas eléctricas*, 5th ed. México, D.F.: McGraw-Hill Interamericana, 2012, pp. 147 - 151.

[18] Cursos Tesla, *Maquina síncrona*. 2020.

[19] J. Fraile Mora, *Tipos constructivos de máquinas síncronas*. 2003.

[20] S. J. Chapman, *Motor de cd sin escobillas simple y su unidad de control asociada. Las entradas a la unidad de control consisten en una fuente de potencia de cd y una señal proporcional a la posición del rotor actual.* 2012.

[21] Díez, Andrés. (2013). INVESTIGACIÓN, DISEÑO Y PROTOTIPO DE UNA BICICLETA ELÉCTRICA Y TECNOLOGÍAS EMERGENTES EN BATERÍAS. Revista de Investigaciones aplicadas 2011-0413. 8.

[22] M. Root, *The TAB battery book*, 1st ed. New York: McGraw-Hill, 2011, pp. 157-182.

[23] A. Hernández Romero, "Análisis económico de un sistema de almacenamiento para la disminución de desvíos de producción en un parque eólico", Maestría, Universidad de Sevilla, 2016.

[24] Sebastián, V., 2019. *Esta Rueda Convierte En Bicicleta Eléctrica Cualquier Bici Tradicional Siempre Que Estés Dispuesto A Pagar 2000 Euros*. [online] Xataka.com. Available at: <https://www.xataka.com/vehiculos/esta-rueda-convierte-bicicleta-electrica-cualquier-bici-tradicional-estes-dispuesto-a-pagar-2000-euros> [Accessed: 20- enero- 2021].

[25] M. Root, *Comparison of energy densities by weight and volume for selected rechargeable batteries*. 2011.

[26] R. Isermann, Mechatronic systems fundamentals. Oxford: Springer, 2005, pp. 25-29. [Último acceso: 02- noviembre- 2020].

[27] B. DeHoff, D. J. H. Levack y R. E. Rhodes, "The Functional Breakdown Structure (FBS) and Its Relationship to Life Cycle Cost", NASA, 2009.

[28] "Motor Simulator - Tools", *Ebikes.ca*, 2021. [Online]. Available: https://ebikes.ca/tools/simulator.html. [Accessed: 10- enero- 2021].

[29] "MXUS-rueda delantera sin escobillas para Bicicleta eléctrica", *AliExpress*. [Online]. Available: https://es.aliexpress.com/item/33033023562.html?spm=a2g0o.productlist.0.0.6667cd22L4cPcN&algo_pvid=8b43e5e2-33be-40e3-92fa-1c20d9dc602f&algo_expid=8b43e5e2-33be-40e3-92fa-

1c20d9dc602f-0&btsid=0bb0623d16113572957058378e679c&ws_ab_test=searchweb0_0,searchweb201602_,searchweb201603_. [Accessed: 11- enero- 2021].

[30] Moebiusuibeom-en, Llanta tipo Westwood. 2020. [Último acceso: 08 marzo 2021].

[31] Moebiusuibeom-en, Llanta tipo Sprint. 2020. [Último acceso: 08 marzo 2021].

[32] Moebiusuibeom-en, Llanta tipo Endrick. 2020. [Último acceso: 08 marzo 2021].

[33] "Rueda de bicicleta", Es.wikipedia.org, 2020. [Online]. Available: https://es.wikipedia.org/wiki/Rueda_de_bicicleta#cite_ref-7. [Último acceso: 08 marzo 2021].

[34] Tuttobike, 2021. [Online]. Available: https://tuttobike.com/products/bicicleta-montana-rodada-26-18-velocidades-monk-kron. [Último acceso: 28 abril 2021].

[35] "Cargador Portátil Dell Power Bank Pw7015l, 18.000mah", Mercado Libre, 2021. [Online]. Available: https://articulo.mercadolibre.com.mx/MLM-874285622-cargador-portatil-dell-power-bank-pw7015l-18000mah-_JM?matt_tool=98947153&matt_word=&matt_source=google&matt_campaign_id=11206065318&matt_ad_group_id=110325470832&matt_match_type=&matt_network=g&matt_device=c&matt_creative=468194411270&matt_keyword=&matt_ad_position=&matt_ad_type=pla&matt_merchant_id=278681045&matt_product_id=MLM874285622&matt_product_partition_id=409902932397&matt_target_id=pla-409902932397&gclid=CjwKCAjwj6SEBhAOEiwAvFRuKAF1PI7548yXln3pSnjqZiD_K1yQsAqHuHq0xySqhxZAabXgheqDYhoCMLMQAvD_BwE. [Último acceso: 28 abril 2021].

[36] "Correas trapezoidales de suministro industrial", Correas Belsa, 2021. [Online]. Available: https://www.belsaibelsa.com/207-correas. [Último acceso: 28 abril 2021].

[37] "Lámina de Aluminio", Metales Díaz, 2021. [Online]. Available: https://metalesdiaz.com/. [Último acceso: 28 abril 2021].

[38] INRIX, Puntuación Global de Tráfico (INRIX Global Traffic Scorecard), 2018. Available: http://inrix.com/scorecard/. [Último acceso: 29 octubre 2020].

[39] VIGA Online, Available: http://viga.online/index.php. [Último acceso: 31 mayo 2021].

[40] Ingeniería mecánica dinámica, R. C. Hibbeler, Edición 12, 2010. [Último acceso: 31 mayo 2021].

[41] Electrónica de potencia – circuitos, dispositivos y aplicaciones, M. H. Rashid, Edición 3, 2010. [Último acceso: 31 mayo 2021].

[42] “GeoOrbital Wheel, la rueda que ofrece transformar nuestra bici tradicional en eléctrica”, Alvarez Raul, 2016. [Online]. Available: https://www.xataka.com/accesorios/geoorbital-wheel-la-rueda-que-ofrece-transformar-nuestra-bici-tradicional-en-electrica. [Último acceso: 15 junio 2021].

[43] “Tabla de factores de rozamiento del pavimento para neumáticos de goma”, Causa directa – Investigación y reconstrucción de accidentes de tráfico, 2021. [Online]. Available:

https://www.causadirecta.com/especial/calculo-de-velocidades/tablas/tabla-de-factores-de-rozamiento-del-pavimento-para-neumaticos-de-goma. [Último acceso: 15 junio 2021]

[44] “Ranking de ciclociudades 2018”, ITDP México a través de la Estrategia Ciclociudades, 2018. [Último acceso: 15 junio 2021]

APÉNDICES

Apéndice A

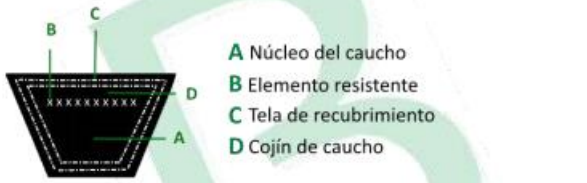
CORREAS DE TRANSMISIÓN

Correas de transmisión trapeciales clásicas, disponibles en perfil Z, A, B, C y D.

Correas industriales forradas con alta resistencia mecánica y buenas condiciones de trabajo a altas y bajas temperaturas (-10º a 90º).

Antiestáticas y resistentes a aceites.

Las correas en V de secciones forradas industriales, están construidas bajo normas internacionales



CORREAS PERFIL C

Ref. correa	Desarrollo exterior (mm)	Desarrollo primitivo (mm)	Desarrollo interior (mm)	Desarrollo interior (")
C33 3/4	945	915	857	33,75
C37 1/2	1041	1011	953	37,5
C38	1053	1023	965	38
C39 3/4	1098	1068	1010	39,75
C40	1104	1074	1016	40
C41	1129	1099	1041	41
C42	1155	1125	1067	42
C43	1180	1150	1092	43
C44	1206	1176	1118	44
C45	1231	1201	1143	45
C46	1256	1226	1168	46
C47	1282	1252	1194	47
C48	1307	1277	1219	48
C49	1333	1303	1245	49
C50	1358	1328	1270	50
C51	1383	1353	1295	51
C52	1409	1379	1321	52
C53	1433	1403	1345	53
C54	1460	1430	1372	54
C55	1485	1455	1397	55
C56	1510	1480	1422	56
C57	1536	1506	1448	57
C58	1561	1531	1473	58
C59	1587	1557	1499	59
C60	1612	1582	1524	60
C61	1637	1607	1549	61
C62	1663	1633	1575	62
C62 1/4	1669	1639	1581	62,25
C63	1688	1658	1600	63
C64	1714	1684	1626	64
C65	1739	1709	1651	65
C66	1764	1734	1676	66

Ref. correa	Desarrollo exterior (mm)	Desarrollo primitivo (mm)	Desarrollo interior (mm)	Desarrollo interior (")
C73	1942	1912	1854	73
C74	1968	1938	1880	74
C75	1993	1963	1905	75
C76	2018	1988	1930	76
C77	2044	2014	1956	77
C78	2069	2039	1981	78
C79	2095	2065	2007	79
C80	2120	2090	2032	80
C81	2145	2115	2057	81
C82	2171	2141	2083	82
C83	2196	2166	2108	83
C84	2222	2192	2134	84
C85	2247	2217	2159	85
C86	2272	2242	2184	86
C87	2298	2268	2210	87
C87 1/2	2311	2281	2223	87,5
C88	2323	2293	2235	88
C89	2349	2319	2261	89
C90	2374	2344	2286	90
C91	2399	2369	2311	91
C92	2425	2395	2337	92
C93	2450	2420	2362	93
C94	2476	2446	2388	94
C95	2501	2471	2413	95
C96	2526	2496	2438	96
C97	2552	2522	2464	97
C97 1/2	2565	2535	2477	97,5
C98	2577	2547	2489	98
C99	2603	2573	2515	99
C100	2628	2598	2540	100
C101	2653	2623	2565	101
C102	2679	2649	2591	102

Apéndice B

MD CUPRUM

ALUMINIO

TABLA 2.1 A — PROPIEDADES MECANICAS

ALEACION Y TEMPLE	RESISTENCIA A LA TENSION Kg./mm2					DUREZA	CORTE Kg./mm2	FATIGA KG./mm2	MODULO
	ESFUERZO DE RUPTURA	PUNTO CEDENTE	ELONGACION porcentaje en 50,8 mm.			BRINELL 500 kg.	ESFUERZO AL CORTE INICIAL	LIMITE DE ENDURE-CIMIENTO	MODULO DE ELASTICIDAD
	MIN.	MAX.	MIN.	Prueba Espesor 1,60 mm.	Prueba Extera Diam. 12,7 mm	ESFERA Diam. 10 mm			ksi x 10 3
1060-0	6	10	2	43	--	19	5	2	10,0
1060-H12	8	11	6	16	--	23	6	3	10,0
1060-H14	9	12	7	12	--	26	7	4	10,0
1060-H16	10	13	8	8	--	30	7	5	10,0
1060-H18	11	---	9	6	--	35	8	5	10,0
1100-0	8	11	2,5	35	45	23	6	4	10,0
1100-H12	10	13	8	12	25	28	7	4	10,0
1100-H14	11	15	10	9	20	32	8	5	10,0
1100-H16	13	17	12	6	17	38	9	6	10,0
1100-H18	15	---	---	5	15	44	9	6	10,0
1350-0	6	10	---	--	--	--	6	---	10,0
1350-H12	8	12	---	--	--	--	6	---	10,0
1350-H14	10	13	---	--	--	--	7	---	10,0
1350-H16	11	15	---	--	--	--	8	---	10,0
1350-H19	13	---	---	--	--	--	11	5	10,0
2011-T3	33	---	28	--	15	95	22	13	10,2
2011-T8	41	---	32	--	12	100	25	13	10,2
2014-0	---	22	11	--	18	45	13	9	10,6
2014-T4, T 451	38	---	21	--	20	105	27	14	10,6
2014-T6, T 651	46	---	39	--	13	135	30	13	10,6
2014-0	---	21	10	21	--	--	13	---	10,5
2014-T3	39	---	24	20	--	--	26	---	10,5
2014 - T4, T451	41	---	25	22	--	--	26	---	10,5
2014 - T6, T651	44	---	39	10	--	--	29	---	10,5
2017-0	---	25	---	--	22	45	13	9	10,5
2017-T4, T451	39	---	23	--	22	105	27	13	10,5
2018-T61	43	---	33	--	12	120	27	12	10,8
2024-0	---	23	11	20	22	47	13	9	10,6
2024-T3	45	---	30	18	--	120	29	14	10,6
2024-T4, T351	45	---	28	20	19	120	29	14	10,6
2024-T361 7 ○	51	---	28	13	--	130	30	13	10,6
2024-0	---	21	10	20	--	--	13	---	10,6
2024-T3	42	---	27	18	--	--	28	---	10,6
2024-T4, T351	41	---	25	19	--	--	28	---	10,6
2024-T361 7 ○	45	---	34	11	--	--	29	---	10,6
2024-T81, T851	46	---	39	6	--	--	28	---	10,6
2024-T861 7 ○	49	---	45	6	--	--	30	---	10,6
2025-T6	41	---	26	--	19	110	25	13	10,4
2036-T4	35	---	20	24	--	--	---	13	10,3
2117-T4	34	---	17	--	27	70	20	10	10,3
2124-T851	49	---	45	--	8	--	---	---	10,6
2218-T72	34	---	26	--	11	95	21	---	10,8

3

MD CUPRUM

ALUMINIO

TABLA 2.1 B — PROPIEDADES MECANICAS

ALEACION Y TEMPLE	RESISTENCIA A LA TENSION Kg./mm2					DUREZA	CORTE Kg./mm2	FATIGA KG./mm2	MODULO
	ESFUERZO DE RUPTURA	PUNTO CEDENTE	ELONGACION porcentaje en 50,8 mm.			BRINELL 500 kg.	ESFUERZO AL CORTE INICIAL	LIMITE DE ENDURE-CIMIENTO	MODULO DE ELASTICIDAD
	MIN.	MAX.	MIN.	Prueba Espesor 1,60 mm.	Prueba Extera Diam. 12,7 mm	ESFERA Diam. 10 mm			ksi x 10
2219-0	---	23	11	18	--	--	--	--	10,6
2219-T42	---	---	---	20	--	--	--	--	10,6
2219-T31,T351	32	---	20	17	--	--	--	--	10,6
2219-T37	34	---	26	11	--	--	--	--	10,6
2219-T62	38	---	25	10	--	--	--	11	10,6
2219-T81, T851	44	---	32	10	--	--	--	11	10,6
2219-T87	45	---	36	10	--	--	--	11	10,6
2618-T61	45	---	38	--	10	115	27	13	10,8
3003-0	10	13	3,5	30	40	28	8	5	10,0
3003-H12	12	16	8	10	20	35	8	6	10,0
3003-H14	14	18	12	8	16	40	10	6	10,0
3003-H16	17	21	15	5	14	47	11	7	10,0
3003-H18	19	---	17	4	10	55	11	7	10,0
3003-0	9	13	3	30	40	--	8	---	10,0
3003-H12	11	15	8	10	20	--	8	---	10,0
3003-H14	13	18	11	8	16	--	10	---	10,0
3003-H16	16	20	14	5	14	--	11	---	10,0
3003-H18	18	---	---	4	10	--	11	---	10,0
3004-0	15	20	8	20	25	45	11	10	10,0
3004-H32	20	25	15	10	17	52	12	11	10,0
3004-H34	23	27	18	9	12	63	13	11	10,0
3004-H36	25	29	20	5	9	70	14	11	10,0
3004-H38	27	---	22	5	6	77	15	11	10,0
3004-0	15	20	6	20	25	--	11	---	10,0
3004-H32	19	24	14	10	17	--	12	---	10,0
3004-H34	22	26	17	9	12	--	13	---	10,0
3004-H36	24	28	19	5	9	--	14	---	10,0
3004-H38	26	---	---	5	6	--	15	---	10,0
3105-0	10	15	4	24	--	--	8	---	10,0
3105-H12	13	18	11	7	--	--	10	---	10,0
3105-H14	15	20	13	5	--	--	11	---	10,0
3105-H16	18	23	15	4	--	--	11	---	10,0
3105-H18	20	---	17	3	--	--	12	---	10,0
3105-H25	16	---	13	8	--	--	11	---	10,0
4032-T6	39	---	33	--	9	120	27	---	11,4
5005-0	11	15	4	25	--	28	8	---	10,0
5005-H12	13	17	10	10	--	--	10	---	10,0
5005-H14	15	19	12	6	--	--	10	---	10,0
5005-H16	17	21	14	5	--	--	11	---	10,0
5005-H18	19	---	---	4	--	--	11	---	10,0
5005-H32	12	16	8	11	--	36	10	---	10,0
5005-H34	14	18	11	8	--	41	10	---	10,0
5005-H36	16	20	13	6	--	46	11	---	10,0
5005-H38	18	---	---	5	--	51	11	---	10,0
5050-0	13	17	4	24	--	36	11	8	10,0
5050-H32	15	20	11	9	--	46	12	9	10,0
5050-H34	18	22	14	8	--	53	13	9	10,0
5050-H36	19	23	15	7	--	58	13	10	10,0
5050-H38	20	---	---	6	--	63	14	10	10,0

4

MD CUPRUM

ALUMINIO

TABLA 2.1 C — PROPIEDADES MECANICAS

ALEACION Y TEMPLE	RESISTENCIA A LA TENSION Kg./mm2					DUREZA	CORTE Kg./mm2	FATIGA KG./mm2	MODULO
	ESFUERZO DE RUPTURA	PUNTO CEDENTE	ELONGACION porcentaje en 50,8 mm.			BRINELL 500 kg.	ESFUERZO AL CORTE INICIAL	LIMITE DE ENDURE-CIMIENTO	MODULO DE ELASTICIDAD
	MIN.	MAX.	MIN.	Prueba Espesor 1,60 mm.	Prueba Extera Diam. 12,7 mm	ESFERA Diam. 10 mm			ksi x 10
5052-0	18	22	7	25	30	47	13	11	10,2
5052-H32	22	27	16	12	18	60	14	12	10,2
5052-H34	24	29	18	10	14	68	15	13	10,2
5052-H36	26	31	20	8	10	73	16	13	10,2
5052-H38	27	---	---	7	8	77	17	14	10,2
5056-0	30	---	15	--	35	65	18	14	10,3
5056-H18	44	---	41	--	10	105	24	15	10,3
5056-H38	42	---	35	--	15	100	23	15	10,3
5083-0	27	35	12	--	22	--	18	---	10,3
5083-H321,H116	31	39	39	--	16	--	---	16	10,3
5086-0	25	31	10	22	--	--	16	---	10,3
5086-H32, H116	28	33	20	12	--	--	---	---	10,3
5086-H34	31	36	24	10	--	--	19	---	10,3
5086-H112	25	---	10	14	--	--	---	---	10,3
5154-0	21	29	8	27	--	58	15	12	10,2
5154-H32	25	30	18	15	--	67	15	13	10,2
5154-H34	27	32	20	13	--	73	17	13	10,2
5154-H36	30	34	23	12	--	78	18	14	10,2
5154-H38	32	---	25	10	--	80	20	15	10,2
5154-H112	21	---	8	25	--	63	---	12	10,2
5252-H25	22	27	---	11	--	68	15	---	10,0
5252-H38, H28	27	---	---	5	--	75	16	---	10,0
5254-0	21	29	8	27	--	58	15	12	10,2
5254-H32	25	30	18	15	--	67	15	13	10,2
5254-H34	27	32	20	13	--	73	17	13	10,2
5254-H36	30	34	23	12	--	78	18	14	10,2
5254-H38	32	---	25	10	--	80	20	15	10,2
5254-H112	21	---	8	25	--	63	---	12	10,2
5454-0	22	29	8	22	--	62	16	---	10,2
5454-H32	25	31	18	10	--	73	17	---	10,2
5454-H34	27	33	20	10	--	81	18	---	10,2
5454-H111	22	---	12	14	--	70	16	---	10,2
5454-H112	22	---	8	18	--	62	16	---	10,2
5456-0	28	37	12	--	24	--	---	---	10,3
5456-H112	29	---	13	--	22	--	---	---	10,3
5456-H321, H116	31	39	22	--	16	90	21	---	10,3
5457-0	11	15	4	22	--	32	8	---	10,0
5457-H25	18	16	16	12	--	48	11	---	10,0
5457-H38, H28	21	19	4	6	--	55	13	---	10,0
5652-0	18	22	7	25	30	47	13	11	10,2
5652-H32	22	27	16	12	18	60	14	12	10,2
5652-H34	24	29	18	10	14	68	15	13	10,2
5652-H36	26	31	20	8	10	73	16	13	10,2
5652-H38	27	---	23	7	8	77	17	14	10,2
5657-H25	14	20	---	12	--	40	8	---	10,0
5657-H38, H28	15	21	---	7	--	50	11	---	10,0

5

MD CUPRUM

ALUMINIO

TABLA 2.1 D — PROPIEDADES MECANICAS

ALEACION Y TEMPLE	RESISTENCIA A LA TENSION Kg./mm2					DUREZA	CORTE Kg./mm2	FATIGA KG./mm2	MODULO
	ESFUERZO DE RUPTURA	PUNTO CEDENTE	ELONGACION porcentaje en 50,8 mm.			BRINELL 500 kg.	ESFUERZO AL CORTE INICIAL	LIMITE DE ENDURE-CIMIENTO	MODULO DE ELASTICIDAD
	MIN.	MAX.	MIN.	Prueba Espesor 1,60 mm.	Prueba Extera Diam. 12,7 mm	ESFERA Diam. 10 mm			ksi x 10
6061-0	---	15	8	25	30	30	8	6	10,0
6061-T4, 7451	21	---	11	22	25	65	17	10	10,0
6061-T6, T651	30	---	25	12	17	95	21	10	10,0
6061-0	---	14	8	25	--	--	8	---	10,0
6061-T4, 7451	19	---	10	22	--	--	15	---	10,0
6061-T6, T651	30	---	25	12	--	--	19	---	10,0
6063-0	9	5	---	--	--	25	7	6	10,0
6063-T1	15	9	14	20	--	42	10	6	10,0
6063-T4	18	9	15	22	--	--	---	---	10,0
6063-T5	19	15	8	12	--	60	12	7	10,0
6063-T6	25	22	8	12	--	73	15	7	10,0
6063-T83	26	25	6	9	--	82	15	---	10,0
6063-T831	21	19	7	10	--	70	13	---	10,0
6063-T832	30	27	8	12	--	95	19	---	10,0
6066-0	15	---	8	--	18	43	10	---	10,0
6066-T4, T451	37	---	21	--	18	90	20	---	10,0
6066-T6, T651	40	---	37	--	12	120	24	11	10,0
6070-T9	39	---	36	10	--	--	24	10	10,0
6101-H111	10	---	8	--	--	--	---	---	10,0
6101-T6	23	---	20	15 ①	--	71	14	---	10,0
6262-T9	41	---	39	--	10	120	25	9	10,0
6351-T4	25	---	15	20	--	--	---	---	10,0
6351-T6	32	---	29	14	--	95	20	---	10,0
6463-T1	15	---	9	20	--	42	10	7	10,0
6463-T5	19	---	15	12	--	60	12	7	10,0
6463-T6	25	---	22	12	--	74	15	7	10,0
7049-T73	53	---	46	--	12	135	31	---	10,4
7049-T7352	53	---	44	--	11	135	30	---	10,4
7050-73510, T73511	---	---	---				---	---	
	51	---	44	--	12	--	---	---	10,4
7050-T7450 10	54	---	48	--	11	--	31	---	10,4
7050-T7651	56	---	50	--	11	--	33	---	10,4
7075-0	23	---	11	17	16	60	15	---	10,4
7075-T6, T651	58	---	51	11	11	150	32	16	10,4
7075-0	22	---	10	17	--	--	15	---	10,4
7075-T6, T651	54	---	47	11	--	--	22	---	10,4
7175-T74	54	---	46	--	11	135	30	16	10,4
7178-0	23	---	11	15	16	--	---	---	10,4
7178-T6, T651	62	---	55	10	11	--	---	---	10,4
7178-T76, T7651	58	---	51	--	11	--	---	---	10,3
7178-0	23	---	10	16	--	--	---	---	10,4
7178-T6, T651	57	---	50	10	--	--	---	---	10,4
7475-T61	58	---	50	11	--	--	---	---	10,2
7475-T651	60	---	52	--	13	--	---	---	10,4
7475-T7351	51	---	43	--	13	--	---	---	10,4
7475-T761	53	---	46	12	--	--	---	---	10,2
7475-T7651	54	---	47	--	12	--	---	---	10,4
7475-T61	53	---	46	11	--	--	---	---	10,2
7475-T651	50	---	43	12	--	--	---	---	10,2
8176-H24	12	---	10	15	--	--	7	---	10,0

6

Apéndice C

ALUMINIO MD CUPRUM

LAMINA EN HOJA LISA

ALEACION 1100

TEMPLE H-14 SEMI-DURO
EN LAMINA

TEMPLE F
EN PLACA

NUM. DE CAT.	CALIBRE			DIMENSIONES		PESO APROXIMADO	
	B. W. G.	MILIMETROS	PULGS.	METROS	PIES	POR HOJA	POR M2
259		25.4	1.000	0.91 x 2.44	3 x 8	153.47	68.830
262		19.0	0.750	0.91 x 2.44	3 x 8	115.108	51.630
2262		19.0	3/4	0.91 x 3.05	3 x 10	143.5	51.630
295		12.70	0.500	0.91 x 2.44	3 x 8	76.762	34.420
2295		12.70	1/2	0.91 x 3.05	3 x 10	95.80	34.420
296		9.53	0.375	0.91 x 2.44	3 x 8	57.560	25.810
2296		9.50	3/8	0.91 x 3.05	3 x 10	71.8	25.810
297	4	6.05	.238	0.91 X 2.44	3 X 8	36.530	16.380
298				0.91 X 3.05	3 X 10	45.663	
299				1.22 X 3.05	4 X 10	60.950	
300				1.22 X 3.66	4 X 12	73.140	
301	6	5.16	.203	0.91 X 2.44	3 X 8	31.155	13.970
302				0.91 X 3.05	3 X 10	38.944	
303				1.22 X 3.05	4 X 10	51.982	
304				1.22 X 3.66	4 X 12	62.379	
306	8	4.19	.165	0.91 X 2.44	3 X 8	25.335	11.360
307				0.91 X 3.05	3 X 10	31.668	
308				1.22 X 3.05	4 X 10	42.271	
309				1.22 X 3.66	4 X 12	50.725	
311	10	3.40	.134	0.91 X 2.44	3 X 8	20.562	9.220
312				0.91 X 3.05	3 X 10	25.703	
281				1.22 X 2.44	4 X 8	27.475	
313				1.22 X 3.05	4 X 10	34.308	
314				1.22 X 3.66	4 X 12	41.169	
315	12	2.77	.109	0.91 X 2.44	3 X 8	16.726	7.500
316				0.91 X 3.05	3 X 10	20.908	
352				1.22 X 2.44	4 X 8	22.350	
317				1.22 X 3.05	4 X 10	27.908	
318				1.22 X 3.66	4 X 12	33.489	
319	14	2.11	.083	0.91 X 2.44	3 X 8	12.734	5.710
320				0.91 X 3.05	3 X 10	15.918	
358				1.22 X 2.44	4 X 8	17.016	
321				1.22 X 3.05	4 X 10	21.247	
322				1.22 X 3.66	4 X 12	25.496	
323	16	1.65	.065	0.91 X 2.44	3 X 8	9.969	4.470
324				0.91 X 3.05	3 X 10	12.461	
430				1.22 X 2.44	4 X 8	13.320	
325				1.22 X 3.05	4 X 10	16.633	
326				1.22 X 3.66	4 X 12	19.959	
327	18	1.25	.049	0.91 X 2.44	3 X 8	7.516	3.370
328				0.91 X 3.05	3 X 10	9.395	
344				1.22 X 2.44	4 X 8	10.050	
329				1.22 X 3.05	4 X 10	12.540	
330				1.22 X 3.66	4 X 12	15.048	

23

LARGO STANDARD
3.66 M. (12 PIES)

BARRAS

CUADRADA

ALEACION 6061 T-6

NUM. DE CAT.	MEDIDAS		PESO APROX. METRO
	MILIMETROS	PULGADAS	
14000	6,4	1/4	0,109
14001	7,9	5/16	0,171
1558	9,5	3/8	0,246
14003	11,1	7/16	0,335
1562	12,7	1/2	0,437
14005	15,9	5/8	0,683
1570	19,0	3/4	0,983
14007	22,2	7/8	1,393
1578	25,4	1	1,748
14009	31,7	1 1/4	2,731
1586	38,1	1 1/2	3,934
1594	50,8	2	6,993
14029	76,2	3	15,735

HEXAGONAL

ALEACION 6061 T-6

NUM. DE CAT.	MEDIDAS		PESO APROX. METRO
	MILIMETROS	PULGADAS	
1602	4,8	3/16	0,056
16020	6,4	1/4	0,095
16000	7,9	5/16	0,147
16001	9,5	3/8	0,213
16002	11,1	7/16	0,290
1612	12,7	1/2	0,378
16004	15,9	5/8	0,591
1620	19,0	3/4	0,853
16007	22,2	7/8	1,156
1628	25,4	1	1,512
1630	28,6	1 1/8	1,914
16010	31,7	1 1/4	2,364
16011	38,1	1 1/2	3,404
1644	50,8	2	6,058
16030	76,2	3	13,627

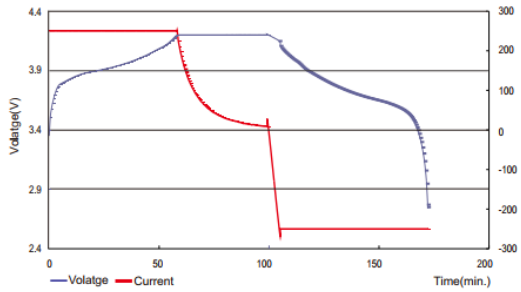
REDONDA

ALEACION 6061 T-6

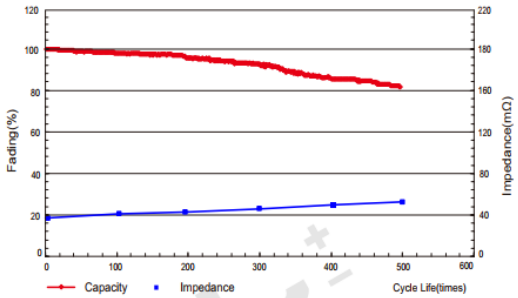
NUM. DE CAT.	DIAMETRO		PESO APROX. METRO
	MILIMETROS	PULGADAS	
1500	3,2	1/8	0,021
1501	4,0	5/32	0,034
13000	4,8	3/16	0,050
13002	6,4	1/4	0,086
1506	7,9	5/16	0,134
13008	9,5	3/8	0,193
1510	11,1	7/16	0,263
13011	12,7	1/2	0,343
13012	14,3	9/16	0,434
1516	15,9	5/8	0,536
1520	19,0	3/4	0,772
1524	22,2	7/8	1,051
1528	25,4	1	1,373
1530	28,6	1 1/8	1,738
1532	31,7	1 1/4	2,146
1534	34,9	1 3/8	2,596
1536	38,1	1 1/2	3,089
1540	44,4	1 3/4	4,205
1544	50,8	2	5,492
1548	63,5	2 1/2	8,582
1878	76,2	3	12,359
1883	88,9	3 1/2	16,281
1886	101,6	4	21,970
1888	114,3	4 1/2	27,807
1891	127,0	5	34,329
1894	152,4	6	49,434
13061	177,8	7	67,573
1897	203,2	8	87,883

Apéndice D

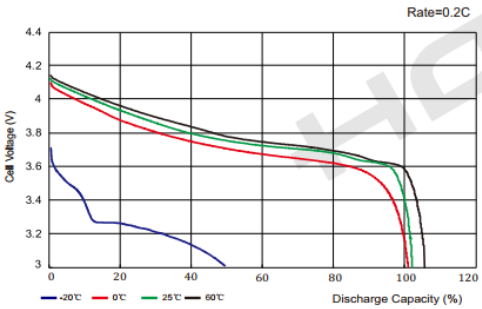
Charge & Discharge (25°C)



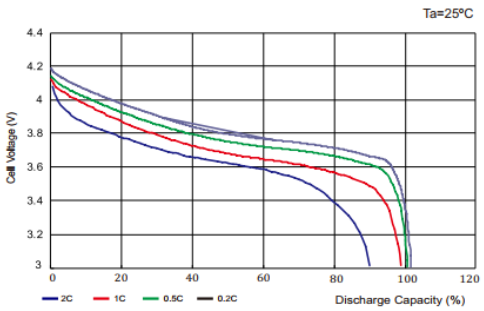
Cycle Life



Discharge Characteristics(Tempertature)



Discharge Characteristics (Rate)



ANEXOS

Anexo 1

Código documentado en lenguaje Micropython para la gestión del sistema de captación y acondicionamiento de energía eléctrica y habilitación de la conexión Bluetooth por medio del microcontrolador ESP32

Clase Main del ESP32

```
from machine import Pin, ADC, PWM
from time import sleep_ms
from bluet import BLE
pwm = 0
# ================= PWM ============================================
def trabajar(_duty):
    trabajo = PWM(Pin(25))
    trabajo.freq(20)                # Establecer frecuencia 50000
    trabajo.duty(int(_duty))        # Establecer duty

# ================= Control del ciclo de trabajo ====================
def control_trabajo(referencia, lectura):
    if referencia > lectura:
        pwm = pwm+1;
    elif referencia < lectura:
        pwm = pwm-1;
    trabajar(pwm)

# ================ main() ==========================================
def main():
    # red led = Pin(2, Pin.OUT)     # LED
    ble = BLE("Enerdrais")          # Creando objeto bluetooth
    adc = ADC(Pin(34))              # Objeto adc de lectura de bicicleta
    adc.atten(ADC.ATTN_11DB)        # Config el rango de voltaje más amplio
    adc2 = ADC(Pin(35))             # Objeto adc de lectura de powerbank
    adc2.atten(ADC.ATTN_11DB)       # Config el rango de voltaje más amplio
    trabajar(0)
    while 1:
        da = ble.get mensaje()      # Getter de comando (Bluetooth)
        if da == "bici":
            while 1:
                val = adc.read()            # Lectura de voltaje
                val = [val, "bici"]         # Estructurar datos
                ble.send(str(val))          # Envio de datos
                control_trabajo(1023, val) # Comparador de pwm y voltaje
                da = ble.get mensaje()      # Parar?
                if da == "stop":
                    trabajar(0)
                    break
                sleep ms(500)
        elif da == "power":
            while 1:
                val = adc2.read()
                val = [val, "powerbank"]
                ble.send(str(val))
                control trabajo(512, val)
                da = ble.get mensaje()
                if da == "stop":
                    trabajar(0)
                    break
                sleep_ms(500)

    main()
```

Clase Bluetooth del ESP32

```
import ubluetooth
from machine import Pin, Timer, ADC
from time import sleep_ms
class BLE:
    def __init__(self, name):
        self.mensa = []
        self.name = name                # Nombre
        self.ble = ubluetooth.BLE()     # Creamos un objeto bluetooth
        self.ble.active(True)           # Activa el bluetooth
        self.ble.irq(self.ble irq)      #
        self.led = Pin(2, Pin.OUT)      # Led azul como salida
        self.timer1 = Timer(0)          # Tiempos
        self.timer2 = Timer(1)          # Tiempos
        self.disconnected()             #
        self.register()
        self.advertiser()

    def connected(self):
        self.timer1.deinit()
        self.timer2.deinit()

    def disconnected(self):
        self.timer1.init(
            period=1000, mode=Timer.PERIODIC, callback=lambda t: self.led(1)
        )
        sleep ms(200)
        self.timer2.init(
            period=1000, mode=Timer.PERIODIC, callback=lambda t: self.led(0)
        )

    def ble_irq(self, event, data):
        adc = ADC(Pin(34))
        if event == 1:
            """ESP32 Conectado"""
            self.connected()
            self.led(1)             # El led se mantiene prendido mientras esta conectado

        elif event == 2:
            """ESP32 Desconectado"""
            self.advertiser()
            self.disconnected()

        elif event == 3:
            """Mensaje recibido"""
            buffer = self.ble.gatts_read(self.rx)
            message = buffer.decode("UTF-8").strip()
            #message = buffer.decode("UTF-8").strip()
            self.mensa = message
            #self.aviso(message)

    def register(self):
        # Nordic UART Service (NUS)
        NUS_UUID = "6E400001-B5A3-F393-E0A9-E50E24DCCA9E" #Identificador unico universal
        RX_UUID  = "6E400002-B5A3-F393-E0A9-E50E24DCCA9E"
        TX_UUID  = "6E400003-B5A3-F393-E0A9-E50E24DCCA9E"

        BLE_NUS = ubluetooth.UUID(NUS_UUID)
        BLE_RX = (ubluetooth.UUID(RX_UUID), ubluetooth.FLAG_WRITE)
        BLE_TX = (ubluetooth.UUID(TX_UUID), ubluetooth.FLAG_NOTIFY)

        BLE_UART = (BLE_NUS, (BLE_TX, BLE_RX))
        SERVICES = (BLE_UART,)
        ((self.tx, self.rx),) = self.ble.gatts_register_services(SERVICES)

    def send(self, data):
        self.ble.gatts_notify(0, self.tx, data + "\n")

    def advertiser(self):
        name = bytes(self.name, "UTF-8")
        self.ble.gap_advertise(
```

```
            100, bytearray("\x02\x01\x02") + bytearray((len(name) + 1, 0x09)) + name
        )

    def get_mensaje(self):
            return self.mensa
```

Anexo 2

Código documentado en lenguaje de programación Kotlin de la aplicación móvil Android para monitorear el tiempo, distancia, velocidad promedio y ruta del usuario

MainActivity

```
package com.androiddevs.runningappyt.ui

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import androidx.navigation.fragment.findNavController
import androidx.navigation.ui.setupWithNavController
import com.androiddevs.runningappyt.R
import com.androiddevs.runningappyt.other.Constants.ACTION_SHOW_TRACKING_FRAGMENT
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.android.synthetic.main.activity_main.*

/*Para probar si funciona la inyección de dependencias usando Dagger Hilt para inyectar la base de
datos en cualquier clase se agrega la anotación @Inject, @AndroidEntryPoint y el código comentado,
que agrega un objeto DAO (Data Access Object) usando el objeto AppModule*/
//@AndroidEntryPoint
/*La anotación @AndroidEntryPoint es necesaria al incluir el menú de navegación porque incluye
fragmentos y por default se quiere inyectar datos en esos fragmentos de la base de datos Room de
SQLite*/
@AndroidEntryPoint
class MainActivity : AppCompatActivity() {
    //@Inject
    //lateinit var runDAO: RunDAO

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        /*Si la aplicación fue cerrada, pero la ruta siguió monitoreandose, es necesario volver a
        la función NavigateToTrackingFragmentIfNeeded declarada debajo, esto pasará si la app fue
        destruida, osea que fue cerrada, pero sino se brincará a la función onNewIntent()*/
        NavigateToTrackingFragmentIfNeeded(intent)

        //Log.d("runDAO", "RUNDAO: ${runDAO.hashCode()}")

        /*Se colocará el funcionamiento del menú inferior y éste seguirá las transiciones dictadas
        en el archivo nav_graph.xml que se encuentra dentro de la carpeta:
        res( o src)/layout/navigation*/
        /*Utilizar el método setSupportActionBar con un objeto toolbar como parámetro le indica a
        Android Studio que encima del proyecto se creó una barra de navegación customizada con un id
        llamado toolbar*/
        setSupportActionBar(toolbar)
        /*Con la siguiente línea de código se especifica que se quiere crear un menú inferior y que
        este debe estar ligado al componente de navegación nav_graph para indicar sus transiciones*/
        bottomNavigationView.setupWithNavController(navHostFragment.findNavController())
        bottomNavigationView.setOnNavigationItemReselectedListener { /*Operación Nula*/ }
```

```
        /*Como se tiene 5 fragmentos, pero solo en 3 de ellos se quiere mostrar el menú de
        navegación, se debe usar el método addOnDestinationChangeList que es de tipo listener al
        fragmento Navhost para indicar en cuales de ellos si se quiere mostrar el menú y en cuales
        no. En específico se mostrará el menú en los fragmentos Recorrido (RunFragment),
        Rendimiento (Statistics Fragment) y Configuración (Settings Fragment), pero no se quiere
        mostrar en los fragmentos de Seguimiento (Tracking Fragment) e Inicio (Setup Fragment)*/
        /*El método se accionará cada que el fragmento donde se encuentra la aplicación cambie,
        además identifica el fragmento de donde viene por medio de su id y ahí es donde se pueden
        añadir condicionales para mostrar el menú inferior o no*/
        navHostFragment.findNavController()
            .addOnDestinationChangedListener { _, destination, _ ->
                when(destination.id){
                    R.id.settingsFragment, R.id.runFragment, R.id.statisticsFragment ->
                        bottomNavigationView.visibility = View.VISIBLE
                    else -> bottomNavigationView.visibility = View.GONE
                }
            }
    }

    /*Función para inicializar la app dependiendo de la forma en la que fue abierta, ya sea si antes
    fue destruida (cerrada a propósito) o si está siendo recién abierta, esto se hace por medio del
    objeto Intent que viene del método .setContentIntent() de la clase TrackingService*/
    override fun onNewIntent(intent: Intent?) {
        super.onNewIntent(intent)
        NavigateToTrackingFragmentIfNeeded(intent)
    }

    /*Función para checar el estado de los comandos mandados hacia la ventana de notificación del
    inicio de servicio de Google Maps, para ver si la notificación está pendiente o fue mandada*/
    private fun NavigateToTrackingFragmentIfNeeded(intent: Intent?){
        if(intent?.action == ACTION_SHOW_TRACKING_FRAGMENT){
            /*Si la notificación fue lanzada por un clic en la ventana de notificación, la
            aplicación navegará hacia el Tracking Fragment, que es el fragmento de seguimiento de
            ruta, esto se hace para que cuando la app sea cerrada, no deje de seguir la ruta y
            que cuando el usuario vuelva a abrir la app, el código vuelva a la función de onCreate*/
            navHostFragment.findNavController().navigate(R.id.action_global_tracking_fragment)
        }
    }
}
```

activity_main

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/rootView"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <com.google.android.material.appbar.AppBarLayout
        android:id="@+id/appBarLayout"
        android:layout_width="match_parent"
        android:layout_height="?android:attr/actionBarSize"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <com.google.android.material.appbar.MaterialToolbar
```

```
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <com.google.android.material.textview.MaterialTextView
            android:id="@+id/tvToolbarTitle"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:text="Enerdrais"
            android:textSize="30sp"
            android:textStyle="bold"
            android:gravity="center_vertical"
            android:textColor="@android:color/white"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintHorizontal_bias="0.5"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

    </com.google.android.material.appbar.MaterialToolbar>
</com.google.android.material.appbar.AppBarLayout>

<!--Este es el layout contenedor para el fragmento principal, los elementos tipo FrameLayout
solo pueden heredar a un solo componente, por lo que se dice que solo pueden tener un elemento
hijo-->
<FrameLayout
    android:id="@+id/flFragment"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    app:layout_constraintBottom_toTopOf="@+id/bottomNavigationView"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/appBarLayout">
    <!--Dentro del FrameLayout es donde se crea el fragmento que contendrá las transiciones
    entre fragmentos-->
    <!--android:name="androidx.navigation.fragment.NavHostFragment" es el atributo más
    importante cuando se trata de navigation components ya que indica cuál es el contenedor del
    menú de navegación, que contendrá los demás componentes de navegación, además se debe
    indicar a Android Studio que ese es el host de navegación poniendo:
    app:defaultNavHost="true" y poniendo app:navGraph="@navigation/nav_graph" para indicar el
    archivo donde se encuentra el diagrama de transiciones-->
    <fragment
        android:id="@+id/navHostFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:name="androidx.navigation.fragment.NavHostFragment"
        app:defaultNavHost="true"
        app:navGraph="@navigation/nav_graph"/>
</FrameLayout>

<!--En la etiqueta de BottomNavigationView se debe asignar el nombre del menú creado para
que a él se pueda relacionar la gráfica de navegación creada en el fragment declarado arriba-->
<com.google.android.material.bottomnavigation.BottomNavigationView
    android:id="@+id/bottomNavigationView"
    android:layout_width="match_parent"
    android:layout_height="56dp"
    app:menu="@menu/bottom_nav_menu"
    app:itemIconTint="@drawable/bottom_nav_selector"
    app:itemTextColor="@drawable/bottom_nav_selector"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toStartOf="parent" />
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

build.gradle (app)

```
//Plugins por default de Android Studio
apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-android-extensions'
/*Este plugin sirve para poder usar la herramienta de Room para la base de datos y demás
librerías importadas con la palabra reservada kapt*/
apply plugin: 'kotlin-kapt'
apply plugin: "androidx.navigation.safeargs.kotlin"
apply plugin: "dagger.hilt.android.plugin"

android {
    compileSdkVersion 29
    buildToolsVersion "29.0.3"

    defaultConfig {
        applicationId "com.androiddevs.runningappyt"
        minSdkVersion 21
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    //Agregado para activar las kotlin-android-extensions
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-
rules.pro'
        }
    }

    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = JavaVersion.VERSION_1_8.toString()
    }
}

dependencies {
    //Librerías default de Android Studio
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.core:core-ktx:1.3.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    testImplementation 'junit:junit:4.13'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'

    //Librerías añadidas
    /*Diseño de Material: Sirve para colocar elementos de diseño como la barra de navegación y la
    librería RecyclerView, el cual facilita que se muestren de manera eficiente grandes conjuntos
    de datos, de forma que no tenga que crear nuevos objetos en la interfaz para mostrar más
    elementos de la lista de datos al hacer scroll.
```

```
Esto se usará para mostrar la velocidad, fecha, duración, etc. de la ruta*/
implementation 'com.google.android.material:material:1.3.0-alpha01'

/*Componentes de Arquitectura View Model: Se diseñó la clase ViewModel con el fin de almacenar
y administrar datos relacionados con la interfaz de manera optimizada. La clase ViewModel
permite que se conserven los datos luego de cambios de configuración, como las rotaciones de
pantallas.*/
implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:2.2.0"

/*Almacenamiento Room: La biblioteca de persistencias Room usa SQLite para el almacenamiento de
datos. La biblioteca ayuda a crear un respaldo de memoria caché de los datos de la app en un
dispositivo que la ejecute. Esta caché permite que los usuarios vean una copia coherente de la
información almacenada en la app, independientemente de si cuentan con conexión a internet o
no.*/
implementation "androidx.room:room-runtime:2.2.5"
kapt "androidx.room:room-compiler:2.2.5"
/*Extensiones de Kotlin y Corrutinas para apoyar el funcionamiento de la base de datos Room*/
implementation "androidx.room:room-ktx:2.2.5"

/*Corrutinas: Una corrutina es un patrón de diseño de simultaneidad que puedes usar en Android
para simplificar el código que se ejecuta de forma asíncrona. Las corrutinas de Kotlin te
permiten escribir código asíncrono, simple y con detenimiento en los más pequeños detalles que
mantiene la capacidad de respuesta de la app mientras administra tareas prolongadas como
operaciones de disco o llamadas de red.*/
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.3.5'
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.3.5'

/*Corrutinas de ciclos de vida: Las corrutinas de Kotlin proporcionan una API que permite
escribir código asíncrono, con ellas, se puede definir un CoroutineScope, lo que ayuda a
administrar cuándo deben ejecutarse las corrutinas. Cada operación asíncrona se ejecuta dentro
de un alcance particular, osea hasta donde puede funcionar.
Se define como LifecycleScope a cada ciclo de vida de un objeto. LifecycleOwner podría ser una
actividad o un fragmento que tenga relacionado un Ciclo de vida. Un Fragmento representa un
comportamiento o una parte de la interfaz de usuario en una Activity. Cualquier corrutina
lanzada en con  alcance se cancela cuando se destruye el ciclo de vida.*/
implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:2.2.0"
implementation "androidx.lifecycle:lifecycle-runtime-ktx:2.2.0"

/*Navigation: Es una librería que permite de una forma más sencilla acceder a las distintas
partes de la aplicación, ya sean layouts, fragments o Activities ,la navegación se refiere a las
interacciones que permiten a los usuarios navegar a través, dentro y fuera de las diferentes
piezas de la app, usando desde simples clics de botones hasta patrones más complejos, como
barras laterales de navegación.*/
implementation "androidx.navigation:navigation-fragment-ktx:2.3.0"
implementation "androidx.navigation:navigation-ui-ktx:2.3.0"

/*Glide: Glide es una librería de código abierto para Android que permite cargar imágenes,
videos y GIFs animados. En la app se utilizará esta librería para tomar screen shots de cada
ruta terminada, para poder mostrarle al usuario todas las rutas que ha hecho.*/
implementation 'com.github.bumptech.glide:glide:4.11.0'
kapt 'com.github.bumptech.glide:compiler:4.11.0'

/*Datos de ubicación - Google Maps location services: Permite saber la ubicación actual del
dispositivo, la dirección y el método de movimiento, y si el dispositivo cruzó un límite
geográfico predefinido, utilizando la API de Google Play Services*/
implementation 'com.google.android.gms:play-services-location:17.0.0'
implementation 'com.google.android.gms:play-services-maps:17.0.0'

/*Dagger: La inserción manual de dependencias o localizadores de servicios en una app para
Android puede ser problemática según el tamaño del proyecto. Dagger sirve para administrar
dependencias, osea las librerías o APIs que utiliza el proyecto.
Dagger genera automáticamente un código que imita el código que se habría escrito a mano para
utilizar una librería. El código se genera en tiempo de compilación.*/
```

```
    implementation "com.google.dagger:dagger:2.28.1"
    kapt "com.google.dagger:dagger-compiler:2.25.2"
    // Dagger Android
    api 'com.google.dagger:dagger-android:2.28.1'
    api 'com.google.dagger:dagger-android-support:2.28.1'
    kapt 'com.google.dagger:dagger-android-processor:2.23.2'
    //Activity KTX for viewModels()
    implementation "androidx.activity:activity-ktx:1.1.0"
    //Dagger - Hilt: Versión más nueva de Dagger
    implementation "com.google.dagger:hilt-android:2.28-alpha"
    kapt "com.google.dagger:hilt-android-compiler:2.28-alpha"
    implementation "androidx.hilt:hilt-lifecycle-viewmodel:1.0.0-alpha01"
    kapt "androidx.hilt:hilt-compiler:1.0.0-alpha01"

    /*Easy permissions: Cada app para Android necesita usar recursos o información ajenos a ella
    misma, como información de la cámara, sensores de movimiento, etc. Se puede declarar un permiso
    y configurar una solicitud de permiso que proporcione el acceso al recurso necesario.
    La librería Easy permissions facilita el acceso a permisos de ubicación.*/
    implementation 'pub.devrel:easypermissions:3.0.0'

    /*Timber: La librería Timber sirve para que el usuario pueda registrarse en la aplicación, osea
    hacer su Login de una forma más sencilla.*/
    implementation 'com.jakewharton.timber:timber:4.7.1'

    /*MPAndroid Chart: MPAndroidChart es una biblioteca de código abiertoque sirve para crear
    gráficas en aplicaciones Android. MPAndroidChart no solo puede dibujar varios gráficos
    estadísticos en dispositivos Android, sino también arrastrar y hacer zoom en los gráficos.*/
    implementation 'com.github.PhilJay:MPAndroidChart:v3.1.0'

    /*Ciclos de vida optimizados: Los componentes optimizados para ciclos de vida realizan acciones
    como respuesta a un cambio en el estado del ciclo de vida de otro componente, como actividades
    o fragmentos. Estos componentes te ayudan a crear un código mejor organizado, y a menudo más
    liviano, que resulta más fácil de mantener y que permite que la aplicación sea más rápida.*/
    implementation 'android.arch.lifecycle:extensions:1.1.1'
}
```

Strings

```
<resources>
    <string name="app_name">RunningApp</string>
    <string-array name="filter_options">
        <item>Fecha</item>
        <item>Tiempo del recorrido</item>
        <item>Distancia</item>
        <item>Velocidad promedio</item>
        <item>Calorías quemadas</item>
    </string-array>
    <string name="google_maps_key" templateMergeStrategy="preserve"
translatable="false">AIzaSyBoIKZqrDXYt1F4JG7oiVW6vp2tTGwnsQY</string>
</resources>
```

AndroidManifest

```
<?xml version="1.0" encoding="utf-8"?>
<!--En el archivo de AndroidManifest es donde se declaran los distintos elementos de hardware a los
que debe acceder la aplicación Android para poder cumplir su función, como sensores, cámara, GPS,
etc-->
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.androiddevs.runningappyt">
```

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<!--android.permission.ACCESS_FINE_LOCATION: Permite que la API de Google Maps determine la
ubicación más precisa posible mediante los proveedores de ubicación disponibles en el
dispositivo móvil, incluido el sistema de posicionamiento global (GPS), así como Wi-Fi y datos
móviles-->
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<!--android.permission.ACCESS_COARSE_LOCATION: Permiso para que la API de Google Maps utilice
Wi-Fi o datos móviles (o ambos) para determinar la ubicación del dispositivo. La API muestra la
ubicación con una exactitud que equivale aproximadamente a una manzana-->
<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
<!--android.permission.ACCESS_BACKGROUND_LOCATION: Sirve para acceder a la ubicación del
dispositivo móvil en segundo plano, mientras la aplicación no está abierta, ya que el usuario
podría bloquear su celular mientras se rastrea su ruta-->
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
<!--android.permission.FOREGROUND_SERVICE: Sirve para poder correr alguna función de la
aplicación en segundo plano-->

<!--Dentro de la etiqueta application se indica el objeto que esté manejando la inyección de
dependencias de la base de datos o demás cosas que se quieran poder inyectar con Dagger Hilt
metiendo en la propiedad name donde se creó el módulo de inyección de dependencias, llamado en
este proyecto como BaseApplication-->
<application
    android:name=".BaseApplication"
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".ui.MainActivity" android:launchMode="singleTask">
        <!--android:launchmode="singleTask" sirve para que solamente una sola tarea de la
        actividad Main va a existir a la vez, osea que se muestre una sola pantalla a la vez
        en la app.
        Las distintas pantallas de la app son instancias de la clase Activity, esto permite que
        solo pueda existir una instancia a la vez en la ejecución de la app-->
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <!--Servicio de Google Maps creado en segundo plano para mostrar y/o usar el mapa en la app.
    La diferencia entre un servicio en segundo plano y en primer plano es que los servicios de
    segundo plano no necesariamente deben mostrar una notificación al usuario, por lo que el
    usuario puede estar usando su dispositivo sin darse cuenta de que un servicio se está
    corriendo, la desventaja de esto es que el servicio puede ser parado por el sistema
    operativo Android por ser un servicio en segundo plano si es que el dispositivo está con
    falta de recursos. Los servicios en primer plano no pueden ser parados por el sistema
    operativo Android y el usuario siempre está consciente de que el servicio está corriendo-->
    <service android:name=".services.TrackingService"
        android:foregroundServiceType="location" />

    <!--La etiqueta de meta datos introducida en este punto sirve para que podamos usar el mapa
     de Google Maps dentro de la aplicación-->
    <!--Esta primera etiqueta de meta datos se pone por default para usar Google Maps-->
    <meta-data
        android:name="com.google.android.gms.version"
        android:value="@integer/google_play_services_version" />
    <!--Para la segunda etiqueta de meta datos se debe colocar la API KEY de Google Maps-->
    <meta-data
        android:name="com.google.android.geo.API_KEY"
```

```
            android:value="@string/google_maps_key" />
    </application>

</manifest>
```

RunAdapter

```
//Plugins por default de Android Studio
apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-android-extensions'
/*Este plugin sirve para poder usar la herramienta de Room para la base de datos y demás
librerías importadas con la palabra reservada kapt*/
apply plugin: 'kotlin-kapt'
apply plugin: "androidx.navigation.safeargs.kotlin"
apply plugin: "dagger.hilt.android.plugin"

android {
    compileSdkVersion 29
    buildToolsVersion "29.0.3"

    defaultConfig {
        applicationId "com.androiddevs.runningappyt"
        minSdkVersion 21
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    //Agregado para activar las kotlin-android-extensions
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-
rules.pro'
        }
    }

    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = JavaVersion.VERSION_1_8.toString()
    }
}

dependencies {
    //Librerías default de Android Studio
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.core:core-ktx:1.3.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    testImplementation 'junit:junit:4.13'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'

    //Librerías añadidas
    /*Diseño de Material: Sirve para colocar elementos de diseño como la barra de navegación y la
```

```
librería RecyclerView, el cual facilita que se muestren de manera eficiente grandes conjuntos
de datos, de forma que no tenga que crear nuevos objetos en la interfaz para mostrar más
elementos de la lista de datos al hacer scroll.
Esto se usará para mostrar la velocidad, fecha, duración, etc. de la ruta*/
implementation 'com.google.android.material:material:1.3.0-alpha01'

/*Componentes de Arquitectura View Model: Se diseñó la clase ViewModel con el fin de almacenar
y administrar datos relacionados con la interfaz de manera optimizada. La clase ViewModel
permite que se conserven los datos luego de cambios de configuración, como las rotaciones de
pantallas.*/
implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:2.2.0"

/*Almacenamiento Room: La biblioteca de persistencias Room usa SQLite para el almacenamiento de
datos. La biblioteca ayuda a crear un respaldo de memoria caché de los datos de la app en un
dispositivo que la ejecute. Esta caché permite que los usuarios vean una copia coherente de la
información almacenada en la app, independientemente de si cuentan con conexión a internet o
no.*/
implementation "androidx.room:room-runtime:2.2.5"
kapt "androidx.room:room-compiler:2.2.5"
/*Extensiones de Kotlin y Corrutinas para apoyar el funcionamiento de la base de datos Room*/
implementation "androidx.room:room-ktx:2.2.5"

/*Corrutinas: Una corrutina es un patrón de diseño de simultaneidad que puedes usar en Android
para simplificar el código que se ejecuta de forma asíncrona. Las corrutinas de Kotlin te
permiten escribir código asíncrono, simple y con detenimiento en los más pequeños detalles que
mantiene la capacidad de respuesta de la app mientras administra tareas prolongadas como
operaciones de disco o llamadas de red.*/
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.3.5'
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.3.5'

/*Corrutinas de ciclos de vida: Las corrutinas de Kotlin proporcionan una API que permite
escribir código asíncrono, con ellas, se puede definir un CoroutineScope, lo que ayuda a
administrar cuándo deben ejecutarse las corrutinas. Cada operación asíncrona se ejecuta dentro
de un alcance particular, osea hasta donde puede funcionar.
Se define como LifecycleScope a cada ciclo de vida de un objeto. LifecycleOwner podría ser una
actividad o un fragmento que tenga relacionado un Ciclo de vida. Un Fragmento representa un
comportamiento o una parte de la interfaz de usuario en una Activity. Cualquier corrutina
lanzada en con  alcance se cancela cuando se destruye el ciclo de vida.*/
implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:2.2.0"
implementation "androidx.lifecycle:lifecycle-runtime-ktx:2.2.0"

/*Navigation: Es una librería que permite de una forma más sencilla acceder a las distintas
partes de la aplicación, ya sean layouts, fragments o Activities ,la navegación se refiere a las
interacciones que permiten a los usuarios navegar a través, dentro y fuera de las diferentes
piezas de la app, usando desde simples clics de botones hasta patrones más complejos, como
barras laterales de navegación.*/
implementation "androidx.navigation:navigation-fragment-ktx:2.3.0"
implementation "androidx.navigation:navigation-ui-ktx:2.3.0"

/*Glide: Glide es una librería de código abierto para Android que permite cargar imágenes,
videos y GIFs animados. En la app se utilizará esta librería para tomar screen shots de cada
ruta terminada, para poder mostrarle al usuario todas las rutas que ha hecho.*/
implementation 'com.github.bumptech.glide:glide:4.11.0'
kapt 'com.github.bumptech.glide:compiler:4.11.0'

/*Datos de ubicación - Google Maps location services: Permite saber la ubicación actual del
dispositivo, la dirección y el método de movimiento, y si el dispositivo cruzó un límite
geográfico predefinido, utilizando la API de Google Play Services*/
implementation 'com.google.android.gms:play-services-location:17.0.0'
implementation 'com.google.android.gms:play-services-maps:17.0.0'

/*Dagger: La inserción manual de dependencias o localizadores de servicios en una app para
Android puede ser problemática según el tamaño del proyecto. Dagger sirve para administrar
```

```
    dependencias, osea las librerías o APIs que utiliza el proyecto.
    Dagger genera automáticamente un código que imita el código que se habría escrito a mano para
    utilizar una librería. El código se genera en tiempo de compilación.*/
    implementation "com.google.dagger:dagger:2.28.1"
    kapt "com.google.dagger:dagger-compiler:2.25.2"
    // Dagger Android
    api 'com.google.dagger:dagger-android:2.28.1'
    api 'com.google.dagger:dagger-android-support:2.28.1'
    kapt 'com.google.dagger:dagger-android-processor:2.23.2'
    //Activity KTX for viewModels()
    implementation "androidx.activity:activity-ktx:1.1.0"
    //Dagger - Hilt: Versión más nueva de Dagger
    implementation "com.google.dagger:hilt-android:2.28-alpha"
    kapt "com.google.dagger:hilt-android-compiler:2.28-alpha"
    implementation "androidx.hilt:hilt-lifecycle-viewmodel:1.0.0-alpha01"
    kapt "androidx.hilt:hilt-compiler:1.0.0-alpha01"

    /*Easy permissions: Cada app para Android necesita usar recursos o información ajenos a ella
    misma, como información de la cámara, sensores de movimiento, etc. Se puede declarar un permiso
    y configurar una solicitud de permiso que proporcione el acceso al recurso necesario.
    La librería Easy permissions facilita el acceso a permisos de ubicación.*/
    implementation 'pub.devrel:easypermissions:3.0.0'

    /*Timber: La librería Timber sirve para que el usuario pueda registrarse en la aplicación, osea
    hacer su Login de una forma más sencilla.*/
    implementation 'com.jakewharton.timber:timber:4.7.1'

    /*MPAndroid Chart: MPAndroidChart es una biblioteca de código abiertoque sirve para crear
    gráficas en aplicaciones Android. MPAndroidChart no solo puede dibujar varios gráficos
    estadísticos en dispositivos Android, sino también arrastrar y hacer zoom en los gráficos.*/
    implementation 'com.github.PhilJay:MPAndroidChart:v3.1.0'

    /*Ciclos de vida optimizados: Los componentes optimizados para ciclos de vida realizan acciones
    como respuesta a un cambio en el estado del ciclo de vida de otro componente, como actividades
    o fragmentos. Estos componentes te ayudan a crear un código mejor organizado, y a menudo más
    liviano, que resulta más fácil de mantener y que permite que la aplicación sea más rápida.*/
    implementation 'android.arch.lifecycle:extensions:1.1.1'
}
```

Converters

```
package com.androiddevs.runningappyt.db

import android.graphics.Bitmap
import android.graphics.BitmapFactory
import androidx.room.TypeConverter
import java.io.ByteArrayOutputStream

class Converters {
    /*La siguiente funcion toma como parámetro un tipo de dato Bitmap y retorna un tipo de dato
    ByteArray, que es un tipo de dato mucho más sencillo para que se pueda añadir a la base de datos
    hecha en la clase Run usando la librería Room.
    El tipo de dato ByteArray consiste de una línea de Bits que representan en este caso al tipo de
    dato Bitmap*/
    /*El problema de hacer esto es que esa serie de Bytes en el tipo de dato ByteArray no nos dicen
    nada de la información que se está guardado en ese tipo de dato, por eso es que también habrá
    una función que hará la acción opuesta, tomando los Bytes del tipo de dato ByteArray y
    convertirlo a un tipo de dato Bitmap, de esa manera podrá la aplicación interpretar esos Bytes
    como un tipo de dato Bitmap*/
```

```
    //Función Convertidora de  Bitmap a ByteArray
    @TypeConverter
    fun toBitMap(bytes: ByteArray): Bitmap{
        /*Aquí se interpretan la sere de bytes que le llegan como parámetro a la función para
        convertirlos a un dato tipo Bitmap*/
        return BitmapFactory.decodeByteArray(bytes, 0, bytes.size)
    }

    //Función Convertidora de ByteArray a Bitmap
    @TypeConverter
    fun fromBitmap(bmp: Bitmap): ByteArray{
        val outputStream = ByteArrayOutputStream()
        /*Al comprimir el archivo Bitmap, se indica el formato del archivo convertido y su calidad*/
        bmp.compress(Bitmap.CompressFormat.PNG, 100, outputStream)
        return outputStream.toByteArray()
    }
}
```

Run

```
package com.androiddevs.runningappyt.db
/*Data class no es más que una clase que sólo contiene estado y no realiza ninguna operación.
Una data class está hecha para que solo contenga atributos que queramos guardar, osea datos*/

/*La clase Run es una entidad de la clase Room, que es una librería importada que se usa en Kotlin
para crear bases de datos*/
import android.graphics.Bitmap
import androidx.room.Entity
import androidx.room.PrimaryKey

/*Dentro del operador Entity se especifica el nombre de la tabla de la base de datos, la entidad
tiene varias columnas y cada una de ellas representa una propiedad de la tabla*/
@Entity(tableName = "running_table")
data class Run(
    /*Valores iniciales de las propiedades de la entidad, las propiedades son todas las columnas
    de la base de datos y la tabla hecha con SQLite dentro del celular se llama entidad*/
    var img: Bitmap? = null,
    var timestamp: Long = 0L,
    var avgSpeedInKNMH: Float = 0f,
    var distanceInMeters: Int = 0,
    var timeInMillis: Long = 0L,
    var caloriesBurned: Int = 0
){
    @PrimaryKey(autoGenerate = true)
    var id: Int? = null
}
```

RunDAO

```
package com.androiddevs.runningappyt.db

import androidx.lifecycle.LiveData
import androidx.room.*

/*El objeto DAO (Data Access Object) es una interfaz que describe todas las posibles acciones que
se puede realizar con la base de datos*/

/*La clase RunDAO tambiém es una entidad de la clase Room pero de tipo DAO que significa Data
Access Object y sirve para obtener los datos requeridos para la base de datos*/
```

```
@Dao
interface RunDAO {
    /*Para poder insertar los nuevos recorridos se usa la anotación @Insert usando el parámetro
    onClonflict, esto significa que cuando se quiera isnertar un recorrido que ya existía en la base
    de datos, esta será reemplazada por la que se añadió después por medio de la suspend function
    realizada por medio de una corrutina*/
    //Insertar  datos a la base de datos
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertRun(run: Run)

    //Borrar datos de la base de datos
    @Delete
    suspend fun deleteRun(run: Run)

    /*Obtener todos los datos de la base de datos en tiempo real, en forma descendente (poniendo los
    nuevos hasta arriba), de la entidad running_table y de la clase Run donde se declaró la base de
    datos*/
    //Fecha
    @Query("SELECT * FROM running_table ORDER BY timestamp DESC")
    fun getAllRunsSortedByDate(): LiveData<List<Run>>
    //Tiempo en milisegundos
    @Query("SELECT * FROM running_table ORDER BY timeInMillis DESC")
    fun getAllRunsSortedByTimeInMillis(): LiveData<List<Run>>
    //Calorías quemadas
    @Query("SELECT * FROM running_table ORDER BY caloriesBurned DESC")
    fun getAllRunsSortedByCaloriesBurned(): LiveData<List<Run>>
    //Velocidad promedio
    @Query("SELECT * FROM running_table ORDER BY avgSpeedInKNMH DESC")
    fun getAllRunsSortedByAvgSpeed(): LiveData<List<Run>>
    //Distancia en metros
    @Query("SELECT * FROM running_table ORDER BY distanceInMeters DESC")
    fun getAllRunsSortedByDistance(): LiveData<List<Run>>

    /*Obtener promedios y totales de los datos deseados, para ello se usan los métodos AVG y SUM de
    la entidad running_table para sus distintas propiedades, osea las columnas de la tabla, para
    ello se indica el tipo de dato adquirido por medio de un Query*/
    //Tiempo total en milisegundos
    @Query("SELECT SUM(timeInMillis) FROM running_table")
    fun getTotalTimeInMillis(): LiveData<Long>
    //Tiempo total en milisegundos
    @Query("SELECT SUM(caloriesBurned) FROM running_table")
    fun getTotalCaloriesBurned(): LiveData<Int>
    //Tiempo total en milisegundos
    @Query("SELECT SUM(distanceInMeters) FROM running_table")
    fun getTotalDistance(): LiveData<Int>
    //Tiempo total en milisegundos
    @Query("SELECT AVG(avgSpeedInKNMH) FROM running_table")
    fun getTotalAvgSpeed(): LiveData<Float>
}
```

RunningDatabase

```
package com.androiddevs.runningappyt.db

import androidx.room.Database
import androidx.room.RoomDatabase
import androidx.room.TypeConverters

/*Como esta es una clase de tipo Database, se usa la anotación @Database para indicar a la librería
Room que esta es la clase que aloja la base de datos y como parámetros de la anotación se indica la
entidad o entidades de la base de datos (tablas de la base de datos) y su versión*/
```

```
@Database(
    entities = [Run::class],
    version = 1
)

/*En adición a la anotación @Database se indica la clase que ejecuta una conversión (si es que la
hay, que en este caso si hay una) usando la anotación @TypeConverters, indicando como su parámetro
la clase que realiza las conversiones*/
@TypeConverters(Converters::class)

/*Las bases de dato hechas con la librería Room se deben crear en una clase abstracta y va a
heredar de la clase RoomDatabase*/
abstract class RunningDatabase : RoomDatabase(){
    /*Lo único que hará esta clase es tener una función abstracta que retorne el objeto RunDAO
    (Data Access Object) creado en una interfaz para obtener los datos de la base de datos.*/
    abstract fun getRunDAO(): RunDAO
}
```

AppModule

```
package com.androiddevs.runningappyt.di
/*Este objeto fue creado para ayudar a Dagger Hilt a la inyección de dependencias en el proyecto*/

import android.content.Context
import android.content.Context.MODE_PRIVATE
import android.content.SharedPreferences
import androidx.room.Room
import com.androiddevs.runningappyt.db.RunningDatabase
import com.androiddevs.runningappyt.other.Constants.KEY_FIRST_TIME_TOGGLE
import com.androiddevs.runningappyt.other.Constants.KEY_NAME
import com.androiddevs.runningappyt.other.Constants.KEY_WEIGHT
import com.androiddevs.runningappyt.other.Constants.RUNNING_DATABASE_NAME
import com.androiddevs.runningappyt.other.Constants.SHARED_PREFERENCES_NAME
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.android.components.ApplicationComponent
import dagger.hilt.android.qualifiers.ApplicationContext
import javax.inject.Singleton

/*Todos los módulos creados para la clase Dagger, deben incluir las anotaciones @Module e
@InstallIn*/
@Module
/*Dentro de la anotación de @InstallIn se pone ApplicationComponent::class para que cuando se creen
objetos en la función onCreate de la clase donde se llamó la clase Dagger Hilt, permitiendo que se
cree la Dependency Injection y que cuando el usuario deje de usar la aplicación, esas dependencias
sean destruidas*/
@InstallIn(ApplicationComponent::class)

object AppModule {
    /*Dentro del objeto se indica la forma en la que se va a poder inyectar la base de datos Room
    a las distintas clases del proyecto que la necesiten*/

    /*Para indicar que la base de datos podrá ser inyectada y proverá datos, se utiliza la anotación
    @Provides para indicar a la librería Dagger Hilt que el resultado de la función se puede usar
    para crear otras dependencias o inyectarse a las clases del proyecto para proveer datos, para
    que la inyección sea de una misma base de datos y no se creen varias instancias a la vez, se usa
    la anotación @Singleton*/
    @Singleton
    @Provides
    fun provideRunningDataBase(
```

```
        /*El contexto en el cual se crea la base de datos se crea como parámetro de la función,
        por medio de un objeto Context llamado app y la anotación @Application context*/
        @ApplicationContext app: Context
    ) = Room.databaseBuilder(
        /*Dentro del método databaseBuilder se usa el objeto Context creado como parámetro de la
        función, la clase donde se encuentra la base de datos y el nombre de la base de datos*/
        app,
        RunningDatabase::class.java,
        RUNNING_DATABASE_NAME
    /*Después de haber indicado la clase que se quiere inyectar por medio de Dagger Hilt, se debe
    usar el método build() para que la base de datos pueda ser inyectada*/
    ).build()

    /*Al dejar la clase sin la anotación @Singleton, lo que pasará es que cuando se quiera injectar
    la base de datos en algunas clases del proyecto, se crearan nuevas instancias de la base de
    datos cada vez, por lo que cada clase o parte del proyecto tendrá su propia instancia de la
    clase distinta de las demás, teniendo así varias bases de datos distintas. Para que no pase eso
    se agregó hasta arriba de la función la anotación @Singleton, la cual hace que la base de datos
    inyectada a las distintas clases del proyecto sea la misma y no se creen varias instancias
    diferentes*/
    /*Se debe repetir el proceso para el objeto DAO (Data Access Object) usando la clase donde fue
    creada la base de datos como el tipo de dato de la función*/
    @Singleton
    @Provides
    fun provideRunDao(db: RunningDatabase) = db.getRunDAO()


    /*Función para indicar que el usuario tiene que introducir el nombre y peso antes de usar la
    aplicación para también inyectarlo en otros fragmentos y que se pueda usar en ellos, esto se
    hará a través de un objeto  y crear una variable llamada firstTimeToggle de tipo Boolean que es
    usada para que el programa sepa si es la primera vez que el usuario entra a la aplicación o no*/
    /*Se crearon claves en el archivo de Constants donde se indicó el nombre de las preferencias
    del usuario, su nombre, peso y si es la primera vez que ingresa a la app*/
    @Singleton
    @Provides
    fun provideSharedPreferences(@ApplicationContext app: Context) =
        app.getSharedPreferences(SHARED_PREFERENCES_NAME, MODE_PRIVATE)

    /*Función que provee el nombre, el peso y la variable Booleana que indica si es la primera vez
    que el usuario ingresa a la aplicación para que se pueda utilizar en distintas partes de la app
    sin necesidad de siempre estar ingresando a las shared preferences*/
    @Singleton
    @Provides
    fun provideName(sharedPref: SharedPreferences) = sharedPref.getString(KEY_NAME, "") ?: ""
    @Singleton
    @Provides
    fun provideWeight(sharedPref: SharedPreferences) = sharedPref.getFloat(KEY_WEIGHT, 80f)
    @Singleton
    @Provides
    fun provideFirstTimeToggle(sharedPref: SharedPreferences) = sharedPref.getBoolean(
        KEY_FIRST_TIME_TOGGLE, true)
}
```

ServiceModule

```
package com.androiddevs.runningappyt.di

import android.app.PendingIntent
import android.content.Context
import android.content.Intent
import androidx.core.app.NotificationCompat
```

```
import com.androiddevs.runningappyt.R
import com.androiddevs.runningappyt.other.Constants
import com.androiddevs.runningappyt.ui.MainActivity
import com.google.android.gms.location.FusedLocationProviderClient
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.android.components.ServiceComponent
import dagger.hilt.android.qualifiers.ApplicationContext
import dagger.hilt.android.scopes.ServiceScoped

/*El objeto ServiceModule guardará todas las dependencias del servicio de rastreo de Google Play y
además manejará el ciclo de vida del mismo servicio de rastreo para que las dependencias dentro del
módulo solo puedan existir durante el servicio de seguimiento de rastreo, cuando este servicio no
esté funcionando las dependencias no existirán, optimizando así los recursos de la aplicación.
El manejo de dependencias se hará a través de la librería Dagger Hilt usando la anotación @Module
e @InstallIn*/

/*La anotación @InstallIn sirve para declarar que tanto pueden durar las dependencias dentro del
módulo, pero las anotaciones de alcance como @Singleton sirven para determinar cuantas instancias
del módulo (que es un objeto) se pueden crear*/
@Module
@InstallIn(ServiceComponent::class)
/*Las dependencias de este módulo no son de tipo @Singleton, esto significa que se pueden instanciar
y existir varias copias de este módulo para usarse en distintas partes de la aplicación y que cada
instancia sea distinta de la otra, además al no ser @Singleton, las dependencias solo existen
mientras el servicio de rastreo esté activo, sino las demás clases no las pueden instanciar*/
object ServiceModule {
    /*La anotación @ServiceScoped indica que mientras el ciclo de vida del servicio exista solo
    habrá una instancia del objeto fusedLocationProviderClient que sirve para ver los cambios de
    ubicación en el recorrido por medio de la API de Google Play Services*/
    @ServiceScoped
    @Provides
    fun provideFusedLocationProviderClient(
        @ApplicationContext app: Context
    ) = FusedLocationProviderClient(app)

    /*Cada módulo tiene su propio alcance (scope) cuando se usa la librería para inyección de
    dependencias Dagger Hilt*/
    /*Para que los comandos (intents) pendientes puedan ser lanzados se debe usar la siguiente
    función que utiliza el método PendingIntent para que por medio de la acción guardada en el
    archivo Contants llamada ACTION_SHOW_TRACKING_FRAGMENT y la bandera de actualización de
    notificaciones pendientes (la cual actualiza el registro de notificaciones pendientes) las
    va lanzando una por una*/
    @ServiceScoped
    @Provides
    fun provideMainActivityPendingIntent(
        @ApplicationContext app: Context
    ) = PendingIntent.getActivity(
        app,
        0,
        Intent(app, MainActivity::class.java). also{
            it.action = Constants.ACTION_SHOW_TRACKING_FRAGMENT
        },
        PendingIntent.FLAG_UPDATE_CURRENT
    )

    /*Dagger Hilt también manejará la creación de la notificación, para ello se le debe
    proporcionar el ID del canal de comunicación creado como constante en el archivo de Constants
    del paquete other.
    - Se le aplica el método .setAutoCancel(false) para prevenir que si el usuario da clic sin
    querer en la notificación, esta desaparezca.
    - Se aplica el método .setOngoing(true) para que la notificación no pueda ser escondida con un
```

```
    movimiento de dedo sobre ella.
    - Se aplica el método .setSmallIcon(R.drawable.ic_directions_run_black_24dp) para que se muestre
    con un ícono en específico que se encuentra dentro de las carpetas del proyecto en
    app/res(o src)/drawable.
    - El método .setContentTitle("Aplicación Enerdrais") aplicado sirve para añadir el título de la
    notificación.
    - El método .setContentText("00:00:00") especifica el texto que aparecerá en la notificación,
    que en este caso se usa para mostrar el tiempo de inicio del recorrido, que cada notificación
    se actualizará para ir aumentando el tiempo del recorrido mostrado en segundos.*/
    @ServiceScoped
    @Provides
    fun provideBaseNotificationBuilder(
        @ApplicationContext app: Context,
        pendingIntent: PendingIntent
    ) = NotificationCompat.Builder(app, Constants.NOTIFICATION_CHANNEL_ID)
        .setAutoCancel(false)
        .setOngoing(true)
        .setSmallIcon(R.drawable.ic_directions_run_black_24dp)
        .setContentTitle("Aplicación Enerdrais")
        .setContentText("00:00:00")
        .setContentIntent(pendingIntent)
}
```

Constants

```
package com.androiddevs.runningappyt.other

import android.graphics.Color

/*Este objeto simplemente se usa para declarar el nombre constante de la base de datos usada en
el módulo AppModule*/

object Constants {
    /*Nombre de la base de datos*/
    const val RUNNING_DATABASE_NAME = "running_db"

    /*Permiso para el seguimiento de la ubicación*/
    const val REQUEST_CODE_LOCATION_PERMISSION = 0

    /*Comandos (o intents) mandados al servicio de Google Maps para empezar, resumir, pausar o
    parar el mapa de Google Maps*/
    //Empezar o resumir el servicio de Google Maps
    const val ACTION_START_OR_RESUME_SERVICE = "ACTION_START_OR_RESUME_SERVICE"
    //Pausar el servicio de Google Maps
    const val ACTION_PAUSE_SERVICE = "ACTION_PAUSE_SERVICE"
    //Detener el servicio de Google Maps
    const val ACTION_STOP_SERVICE = "ACTION_STOP_SERVICE"
    //Se añade una acción extra para mostrar el fragmento de seguimiento de ruta
    const val ACTION_SHOW_TRACKING_FRAGMENT = "ACTION_SHOW_TRACKING_FRAGMENT"

    /*Intervalo de muestreo para el seguimiento de ruta en milisegundos*/
    const val LOCATION_UPDATE_INTERVAL = 5000L
    /*Intervalo de muestreo máximo para el seguimiento de ruta en milisegundos*/
    const val FASTEST_LOCATION_UPDATE_INTERVAL = 2000L

    /*Id y nombre de la ventana de notificación para poder empezar el servicio de seguimiento de
    ruta de Google Maps, además es necesario crear un id numérico para rastrear las notificaciones
    en sí*/
    const val NOTIFICATION_CHANNEL_ID = "tracking_channel"
    const val NOTIFICATION_CHANNEL_NAME = "tracking"
```

```
    const val NOTIFICATION_ID = 1

    /*Color de la línea de seguimiento de ruta*/
    const val POLYLINE_COLOR = Color.RED
    /*Color de la línea de seguimiento de ruta*/
    const val POLYLINE_WIDTH = 8f

    /*Zoom sobre el mapa cuando sigue la línea de seguimiento de ruta*/
    const val MAP_ZOOM = 15f

    /*Tiempo del delay de la corrutina que convierte el tiempo de milisegundos a segundos*/
    const val TIMER_UPDATE_INTERVAL = 50L

    /*Nombre del compartimiento de preferencias y el nombre de la variable que indica si es la
    primera vez que un usuario ingresa a la app o no, además del nombre de las claves donde
    se guardará el nombre y peso del usuario*/
    const val SHARED_PREFERENCES_NAME = "sharedPref"
    const val KEY_FIRST_TIME_TOGGLE = "KEY_FIRST_TIME_TOGGLE"
    const val KEY_NAME = "KEY_NAME"
    const val KEY_WEIGHT = "KEY_WEIGHT"
}
```

CostumMakerView

```
package com.androiddevs.runningappyt.other

import android.content.Context
import com.androiddevs.runningappyt.db.Run
import com.github.mikephil.charting.components.MarkerView
import com.github.mikephil.charting.data.Entry
import com.github.mikephil.charting.highlight.Highlight
import com.github.mikephil.charting.utils.MPPointF
import kotlinx.android.synthetic.main.marker_view.view.*
import java.text.SimpleDateFormat
import java.util.*

class CustomMarkerView(
    val runs: List<Run>,
    c: Context,
    layoutId: Int
) :MarkerView(c, layoutId){

    override fun getOffset(): MPPointF {
        return MPPointF(-width / 2f, -height.toFloat())

    }

    override fun refreshContent(e: Entry?, highlight: Highlight?) {
        super.refreshContent(e, highlight)
        if(e == null){
            return
        }
        val curRunId = e.x.toInt()
        val run = runs[curRunId]
        /*Para mostrar la fecha del recorrido se usará la clase Calendar, que en un inicio fue
            creada en forma de milisegundos pero se transfromará para solo mostrar la fecha en este
            caso*/
        val calendar = Calendar.getInstance() .apply {
            //Timestamp es un tipo de dato que muestra la fecha y hora con milisegundos
            timeInMillis = run.timestamp
        }
```

```
        /*Aquí se indica como parámetro del método SimpleDateFormat el formato en el que se va
        a mostrar la fecha del recorrido poniendo:
        dd(día).MM(mes).yy(año) el punto indica el órden en el que se acomodan, luego se debe
        colocar la zona horaria en el que se va a colocar la hora, para ello se usa el método
        Locale.getDefault() para obtener la zona horaria de los datos del teléfono*/
        val dateFormat = SimpleDateFormat("dd.MM.yy", Locale.getDefault())
        /*Se coloca en la propiedad text del elemento tvDate del fragmento item_run la fecha y
        hora del recorrido*/
        tvDate.text = dateFormat.format(calendar.time)
        /*Se concatena en una variable el valor del dato avgSpeedInKNMH contenido en la base de
        datos para poder mostrarlo en pantalla en la propiedad text del elemento tvAvgSpeed del
        fragmento item_run*/
        val avgSpeed = "${run.avgSpeedInKNMH} km/h"
        tvAvgSpeed.text = avgSpeed
        /*Se concatena en una variable el valor del dato distanceInMeters contenido en la base
        de datos para poder mostrarlo en pantalla en la propiedad text del elemento tvDistance
        del fragmento item_run*/
        val distanceInKm = "${run.distanceInMeters / 1000f} km"
        tvDistance.text = distanceInKm
        /*Se coloca en la propiedad text del elemento tvTime del fragmento item_run el tiempo
        del recorrido obtenido de la función getFormattedStopWatchTime() de la clase
        TrackingUtility al cual se le pasa como parámetro el tiempo en milisegundos obtenido del
        valor timeInMillis obtenido de la base de datos a través del objeto run*/
        tvDuration.text = TrackingUtility.getFormattedStopWatchTime(run.timeInMillis)
        /*Se concatena en una variable el valor del dato caloriesBurned contenido en la base de
        datos para poder mostrarlo en pantalla en la propiedad text del elemento tvCalories del
        fragmento item_run*/
        val caloriesBurned = "${run.caloriesBurned} kcal"
        tvCaloriesBurned.text = caloriesBurned
    }
}
```

SortType

```
package com.androiddevs.runningappyt.other
/*La clase ENUM es una clase que solo contiene constantes enumeradas y se usa mucho para el
manejo de datos*/

enum class SortType {
    DATE, RUNNING_TIME, AVG_SPEED, DISTANCE, CALORIES_BURNED
}
```

TrackingUtility

```
package com.androiddevs.runningappyt.other

import android.Manifest
import android.content.Context
import android.location.Location
import android.os.Build
import com.androiddevs.runningappyt.services.Polyline
import pub.devrel.easypermissions.EasyPermissions
import java.util.concurrent.TimeUnit

/*Este objeto sirve para manejar los permisos necesarios para acceder a la localización del usuario
por medio de GPS, Wifi y otros medios*/

object TrackingUtility {
    /*La siguiente función checa si el sistema operativo es Android o Android Q (que es una
```

```
actualización del sistema operativo Android) para checar si es necesario el permiso
android.permission.ACCESS_BACKGROUND_LOCATION o no utilizando la librería Easy permissions y
si el usuario ya ha concedido el permiso solicitado en el archivo AndroidManifest*/
fun hasLocationPermissions(context: Context) =
    if(Build.VERSION.SDK_INT < Build.VERSION_CODES.Q){
        EasyPermissions.hasPermissions(
            /*En Android normal solo son necesarios los permisos ACCESS_FINE_LOCATION y
            ACCESS_COARSE_LOCATION*/
            context,
            Manifest.permission.ACCESS_FINE_LOCATION,
            Manifest.permission.ACCESS_COARSE_LOCATION
        )
    } else{
        EasyPermissions.hasPermissions(
            /*En Android Q es necesario pedir los permisos ACCESS_FINE_LOCATION,
            ACCESS_COARSE_LOCATION y ACCESS_BACKGROUND_LOCATION*/
            context,
            Manifest.permission.ACCESS_FINE_LOCATION,
            Manifest.permission.ACCESS_COARSE_LOCATION,
            Manifest.permission.ACCESS_BACKGROUND_LOCATION
        )
    }

/*Función que calcula la distancia entre cada polyline que es una lista de listas de las
coordenadas recopiladas en el recorrido del usuario y fue un tipo de dato declarado en la
parte de arriba de este código*/
fun calculatePolyLineLength(polyline: Polyline): Float{
    //Variable que guardará la distancia recorrida por el usuario con valor inicial de 0
    var distance = 0f
    /*Bucle for que recorrerá las posiciones del Polyline creado en el recorrido menos 2 porque
    la distancia será calculada restando las posiciones i e i +1 del tamaño del polyline, por
    eso es que se debe hacer hasta el tamaño de polyline menos 2*/
    for(i in 0..polyline.size-2){
        var pos1 = polyline[i]
        var pos2 = polyline[i + 1]
        /*Array donde se guardarán los resultados de las distancias entre los puntos
        consecuentes del polyline de la ruta*/
        var result = FloatArray(1)
        /*Luego se utilizará el método distanceBetween de la clase Location para medir la
        distancia entre cada uno de los 2 puntos accedidos en el bucle for al cual se lo debo
        pasar en forma de latitud y longitud de sus punto inicial y final, además de
        proporcionar un array de resultados*/
        Location.distanceBetween(
            pos1.latitude,
            pos1.longitude,
            pos2.latitude,
            pos2.longitude,
            result
        )
        distance += result[0]
    }
    return distance
}

/*Función que muestra los valores del tiempo monitoreado y obtenido por medio de la función
TrackingService en una forma que se vea en horas, minutos, segundos y milisegundos, la función
recibe como parámetro los milisegundos de la ruta que son tipo de dato Long y una variable
de tipo boolean, la función retornará un tipo de dato String para mostrar el tiempo en pantalla
*/
fun getFormattedStopWatchTime(ms: Long, includeMillis:Boolean = false):String{
    /*Almacena los milisegundos recibidos en el parámetro en otra variable para manejarla en el
    código*/
    var milliseconds  = ms
```

```
        //Convierte los milisegundos recibidos en el parámetro a horas
        var hours = TimeUnit.MILLISECONDS.toHours(milliseconds)
        /*Se convierten las horas de nuevo en milisegundos para mostrar las horas en forma de String
        y usar los milisegundos restantes que haya cuando le restemos los milisegundos que forman
        una hora para poder crear los minutos del temporizador*/
        milliseconds -= TimeUnit.HOURS.toMillis(hours)
        //Convierte los milisegundos restantes a minutos
        val minutes = TimeUnit.MILLISECONDS.toMinutes(milliseconds)
        /*Se convierten los minutos de nuevo en milisegundos para mostrarlos en forma de String
        y usar los milisegundos restantes que haya cuando le restemos los milisegundos que forman
        un minuto para poder crear los segundos del temporizador*/
        milliseconds -= TimeUnit.MINUTES.toMillis(minutes)
        //Convierte los milisegundos restantes a segundos
        val seconds = TimeUnit.MILLISECONDS.toSeconds(milliseconds)
        /*Si la variable booleana declarada en la función es false, que en un inicio siempre lo es,
        esta retornará un String que contiene las horas, minutos y segundos del recorrido*/
        if(!includeMillis){
            /*Si las horas del recorrido transcurridas son menores a 10 se añadirá un cero antes
            del número de horas mostradas del recorrido y si es mayor a 10 se mostrará el valor de
            las horas transcurridas de forma normal en el temporizador y se hace lo mismo con los
            minutosy segundos*/
            return "${if (hours < 10) "0" else ""}$hours:" +
                    "${if (minutes < 10) "0" else ""}$minutes:" +
                    "${if (seconds < 10) "0" else ""}$seconds:"
        }
        /*Para incluir milisegundos en el temporizador se convierten los segundos de nuevo en
        milisegundos para mostrarlos en forma de String y usar los milisegundos restantes que haya
        cuando le restemos los milisegundos que forman un minuto para poder crear los milisegundos
        del temporizador, además los milisegundos restantes se dividen entre 10 para solo mostrar
        dos dígitos y se retornará el mismo resultado que cuando se creó el temporizador sin
        milisegundos*/
        milliseconds -= TimeUnit.SECONDS.toMillis(seconds)
        milliseconds /= 10
        return "${if (hours < 10) "0" else ""}$hours:" +
                "${if (minutes < 10) "0" else ""}$minutes:" +
                "${if (seconds < 10) "0" else ""}$seconds:" +
                "${if (milliseconds < 10) "0" else ""}$milliseconds:"
    }
}
```

MainRepository

```
//Inyección de dependencias en esta clase
class MainRepository @Inject constructor(
    /*Declaración del objeto runDAO instanciado de la interfaz RunDAO dentro del constructor hecho
    con la anotación @Inject para poder usar los métodos creados ahí para insertar elementos a la
    base de datos*/
    val runDAO: RunDAO
){
    /*En la clase Run es donde se decararon todas las propiedades (columnas) de la entidad (tabla)
    de la base de datos hecha con la librería Room basada en SQLite, esta se usa para inyectar la
    base de datos al repositorio*/
    /*Dentro del cuerpo de la clase actual se llama los métodos de la clase RunDAO y se utilizan
    aquí con una función suspend para insertar y borrar datos a los recorridos*/
    suspend fun insertRun(run: Run) = runDAO.insertRun(run)
    suspend fun deleteRun(run: Run) = runDAO.deleteRun(run)

    /*Estas funciones son utilizadas para llamar los datos de cada corrida creados en la base de
    datos con Room, todos estos métodos fueron declarados en la interfaz RunDAO del paquete db*/
    fun getAllRunsSortedByDate() = runDAO.getAllRunsSortedByDate()
    fun getAllRunsSortedByTimeInMillis() = runDAO.getAllRunsSortedByTimeInMillis()
```

```
    fun getAllRunsSortedByCaloriesBurned() = runDAO.getAllRunsSortedByCaloriesBurned()
    fun getAllRunsSortedByAvgSpeed() = runDAO.getAllRunsSortedByAvgSpeed()
    fun getAllRunsSortedByDistance() = runDAO.getAllRunsSortedByDistance()

    /*Ahora se utilizarán todas las funciones para crear las gráficas que mostrarán los datos
    utilizando los métodos creados en la interfaz RunDAO también*/
    fun getTotalTimeInMillis() = runDAO.getTotalTimeInMillis()
    fun getTotalCaloriesBurned() = runDAO.getTotalCaloriesBurned()
    fun getTotalDistance() = runDAO.getTotalDistance()
    fun getTotalAvgSpeed() = runDAO.getTotalAvgSpeed()
}
```

TrackingService.kt

```
package com.androiddevs.runningappyt.services

import android.annotation.SuppressLint
import android.app.NotificationChannel
import android.app.NotificationManager
import android.app.NotificationManager.IMPORTANCE_LOW
import android.app.PendingIntent
import android.app.PendingIntent.FLAG_UPDATE_CURRENT
import android.content.Context
import android.content.Intent
import android.location.Location
import android.os.Build
import android.os.Looper
import androidx.annotation.RequiresApi
import androidx.core.app.NotificationCompat
import androidx.lifecycle.LifecycleService
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.Observer
import com.androiddevs.runningappyt.R
import com.androiddevs.runningappyt.other.Constants.ACTION_PAUSE_SERVICE
import com.androiddevs.runningappyt.other.Constants.ACTION_SHOW_TRACKING_FRAGMENT
import com.androiddevs.runningappyt.other.Constants.ACTION_START_OR_RESUME_SERVICE
import com.androiddevs.runningappyt.other.Constants.ACTION_STOP_SERVICE
import com.androiddevs.runningappyt.other.Constants.FASTEST_LOCATION_UPDATE_INTERVAL
import com.androiddevs.runningappyt.other.Constants.LOCATION_UPDATE_INTERVAL
import com.androiddevs.runningappyt.other.Constants.NOTIFICATION_CHANNEL_ID
import com.androiddevs.runningappyt.other.Constants.NOTIFICATION_CHANNEL_NAME
import com.androiddevs.runningappyt.other.Constants.NOTIFICATION_ID
import com.androiddevs.runningappyt.other.Constants.TIMER_UPDATE_INTERVAL
import com.androiddevs.runningappyt.other.TrackingUtility
import com.androiddevs.runningappyt.ui.MainActivity
import com.google.android.gms.location.FusedLocationProviderClient
import com.google.android.gms.location.LocationCallback
import com.google.android.gms.location.LocationRequest
import com.google.android.gms.location.LocationRequest.PRIORITY_HIGH_ACCURACY
import com.google.android.gms.location.LocationResult
import com.google.android.gms.maps.model.LatLng
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch
import timber.log.Timber
import javax.inject.Inject

/*La clase heredará de la clase LifeCycleService para manejar los ciclos de vida de los objetos
del servicio de Google Maps creados dentro de la clase con la función de observe*/
```

```
/*La clase maneja la notificación de la aplicación para que cuando se esté haciendo un seguimiento
de ruta aunque se cierre la aplicación no se deje de seguir la ruta y al dar clic en la notificación
se rediriga siempre al seguimiento de la ruta. La notificación no es parte de la pantalla de la app
en sí, sino que sale en la parte de arriba del teléfono.*/

/*Para el seguimiento de la ruta se debe hacer una lista de una lista de coordenadas, para
simplificar este tipo de dato se crea un tipo de dato llamado Polyline*/
typealias Polyline = MutableList<LatLng>
typealias Polylines = MutableList<Polyline>

/*La anotación @AndroidEntryPoint se utiliza para poder inyectar elementos creados con Dagger Hilt
que sirve para manejar inyección de dependencias*/
@AndroidEntryPoint
class TrackingService : LifecycleService(){
    /*Por medio de un dato booleano se informará a la clase si el programa se está iniciando por
    primera vez o si se está reanudando su funcionamiento*/
    var isFirstRun = true

    /*Variable booleana para monitorear si el servicio ha sido detenido o no, que en un inicio
    es false, porque cuando se detenga el servicio se quiere interumpir lo que hace el servicio
    de seguimiento de ruta cuando la variable valga true, el código completo del servicio se
    detiene*/
    var serviceKilled = false

    /*Inyección de dependencias creadas con Dagger Hilt para crear el FusedLocationProviderClient
    y así permitir el rastreo de ubicación, el manejo de comandos (intents) pendientes y la
    creación de la ventana de notificaciones de la app*/
    @Inject

    /*Para poder ver los cambios de ubicación se necesita usar una herramienta llamada
    fusedlocationProviderClient, que se crea como una variable que instancia la clase
    FusedlocationProviderClient*/
    lateinit var fusedLocationProviderClient: FusedLocationProviderClient

    /*Para crear el stopwatch de la aplicación se crearán dos objetos que isnctancían la clase
    MutableLiveData que sirve para recopilar datos en tiempo real hacia la aplicación, uno le
    proporcionará el tiempo del recorrido en segundos y el segundo proporcionará el tiempo del
    recorrido a la aplicación en milisegundos. El tiempo en milisegundos se mostrará dentro de la
    interfaz de la aplicación y en la notificación se mostrará el tiempo en segundos porque no es
    buena práctica actualizar una notificación tan seguido para no gastar de más los recursos
    necesarios del dispositivo móvil*/
    private val timeRunInSeconds = MutableLiveData<Long>()

    /*Variable usada para poder hacer la inyección de dependencias hechas con Dagger Hilt*/
    /*Para poder actualizar el tiempo del recorrido dentro de la ventana de notificaciones se deben
    lanzar nuevas ventanas de notificación con el mismo id, de esa forma se actualiza el contenido
    de la notificación, por eso es que se creó un baseNotificationBuilder, que es la notificación
    inical y contendrá las características de todas las notificaciones que se crearán.*/
    @Inject
    lateinit var baseNotificationBuilder: NotificationCompat.Builder
    lateinit var currentNotificationBuilder : NotificationCompat.Builder
    /*Además se piensa agregar botones para poder pausar y resumir el recorrido por medio de la
    misma ventana de notificaciones y para ello se realiza lo siguiente*/


    /*Para poder ver los cambios en la ruta del usuario se va a crear un objeto MutableLiveData
    dentro de un companion object para que se pueda ver desde fuera de la app, este va a ser de
    tipo boleano para que cada que la ubicación cambie de coordenadas el nuevo valor se guarde en
    la variable isTracking*/
    companion object{
        /*Se crea una variable dentro de la función companion object porque se quieren observar los
        cambios del entorno para manejar las variables contenidas*/
```

```
    /*Aquí se declara la variable que monitoreará el tiempo del recorrido en milisegundos por
    medio de una variable que instancía la clase MutableLiveData usada para recopilar datos del
    dispositivo en tiempo real*/
    val timeRunInMillis = MutableLiveData<Long>()
    /*Variable que detecta si las coordenadas del usuario han cambiado o no con un dato de tipo
    MutableLiveData para recopilar datos en tiempo real*/
    val isTracking = MutableLiveData<Boolean>()
    /*Esta variable va a guardar todas las coordenadas de ubicación de un recorrido en
    específico en una lista de coordenadas que servirá para dibujar las coordenadas en el
    mapa, esto se hace así por si el usuario de la bicicleta para su seguimiento de ruta y
    luego lo vuelve a iniciar despues, que la ruta no se pinte en el mapa durante el tiempo
    en el que no se estuvo haciendo el seguimiento de la ruta. Por lo que en sí esta variable
    debe guardar una lista de listas de coordenadas para que se pueda tener varias líneas de
    rutas. Las listas son de tipo MutableList y las coordenadas de tipo LatLng, este tipo de
    dato fue descrito y creado arriba*/
    val pathPoints = MutableLiveData<Polylines>()
}

/*Función para colocar valores iniciales en las variables isTracking para indicar que en un
inicio no se inicializa el seguimiento de la ruta, la variable pathPoints que pinta la ruta en
el mapa, la variable timeRunInSeconds para dar un valor inicial al temporizador de 0*/
private fun postInitialValues(){
    //VALORES INICIALES
    //Estado del seguimiento de ruta
    isTracking.postValue(false)
    //Línea pintada en el mapa para mostrar la ruta
    pathPoints.postValue(mutableListOf())
    //Temporizador
    timeRunInSeconds.postValue(0L)
    timeRunInMillis.postValue(0L)
}

/*En una función onCreate() se inicializará el objeto fusedlocationProviderClient y llamar la
función postInitialValues*/
override fun onCreate() {
    super.onCreate()
    /*Actualización de la ventana de notificaciones por medio de las variables inyectadas por
    Dagger Hilt llamadas baseNotificationBuilder y currentNotificationBuilder, cuando haya
    pasado cierto tiempo de que se haya inciado el recorrido se actualizará el valor de
    currentNotificationBuilder actualizando así la ventana de notificaciones*/
    currentNotificationBuilder = baseNotificationBuilder

    postInitialValues()
    fusedLocationProviderClient = FusedLocationProviderClient(this)

    /*Esta parte del código observa la ruta dibujada en el recorrido para realizar
    actualizaciones de la lista de listas de coordenadas solamente cuando la posición del
    usuario cambie, utilizando la función updateLocationTracking() para que guarde las
    actualizaciones*/
    isTracking.observe(this, Observer {
        //Función que actualiza la posición del rastreo
        updateLocationTracking(it)
        //Función que actualiza la ventana de notificaciones para mostrar el tiempo de rastreo
        updateNotificationTrackingState(it)
    })
}

/*Función creada para poder cancelar el seguimiento de ruta de un recorrido y borrar todos los
datos de rendimiento*/
private fun killService(){
    /*Al ejecutar la función se regresa al estado de recorrido inicial para que ya no pueda
    ser pausado ni resumido sino que inicie de cero, se detiene el servicio creado por el
    código entero de este archivo TrackingService, se reinician los valores del rendimiento
```

```
    del recorrido y se deja de mostrar la ventana de notificaciones*/
    serviceKilled = true
    isFirstRun = true
    pauseService()
    postInitialValues()
    stopForeground(true)
    stopSelf()
}

/*Esta función se activa siempre que se manda un comando hacia el servicio de Google Maps*/
override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
    /*Los comandos (o también llamados intents) que se mandarán al servicio son: empezar,
    resumir, pausar o parar el servicio de Google Maps*/
    intent?.let {
        when(it.action){
            //Empezar o resumir el servicio de Google Maps
            ACTION_START_OR_RESUME_SERVICE -> {
                /*Cuando el servicio se inicia se utiliza la función startForegroundService()
                que se encuentra debajo, ya que la variable isFirstRun es true, pero aquí se
                cambia su valor a false para que la siguiente vez ya no sea la primera vez que
                se corre el programa*/
                if(isFirstRun){
                    startForegroundService()
                    isFirstRun = false
                }else{
                    /*Se imprime en pantalla con la librería Timber el siguiente mensaje cuando
                    se haya reanudado el recorrido*/
                    Timber.d("Servicio reanudado...")
                    /*Cuando se reanuda dl recorrido se reanuda también el monitoreo del tiempo
                    transcurrido en la ruta con la función startTimer()*/
                    startTimer()
                }
            }
            //Pausar el servicio de Google Maps
            ACTION_PAUSE_SERVICE -> {
                /*Se imprime en pantalla con la librería Timber el siguiente mensaje cuando se
                haya pausado el recorrido*/
                Timber.d("Servicio de rastreo de ruta pausado")
                pauseService()
            }
            //Detener el servicio de Google Maps
            ACTION_STOP_SERVICE -> {
                /*Se imprime en pantalla con la librería Timber el siguiente mensaje cuando se
                haya detenido el recorrido*/
                Timber.d("Servicio de rastreo de ruta detenido")
                /*El servicio completo ejecutado por el código de este archivo TrackingService
                es detenido y los datos de rendimiento borrados*/
                killService()
            }
        }
    }
    return super.onStartCommand(intent, flags, startId)
}

/*Función que empieza el monitoreo del tiempo para el recorrido y empieza el objeto observador
para que pueda registrar los cambios de las variables creadas a partir de la clase
MutableLiveData, para ello es necesaria una variable booleana que indique si el monitoreo de
tiempo ha sido inicializado o no, una que guarde tal cual el tiempo transcurrido, otra que
almacene el tiempo total del recorrido, una que contenga el tiempo inicial con el que se
empezó a hacer el recorrido y finalmente una que guarde el último segundo del recorrido*/
//Estado del monitoreo
private var isTimmerEnabled = false
//Tiempo del transcurso del recorrido
```

```
private var lapTime = 0L
//Tiempo total del recorrido
private var timeRun = 0L
//Tiempo inicial con el que se empezó el recorrido
private var timeStarted = 0L
//Último segundo del recorrido
private var lastSecondTimeStamp = 0L
/*La función que inicializa el monitoreo del tiempo se activa cuando se inicializa o reanuda el
servicio de seguimiento de ruta de la API de Google Play Services por medio del botón de INICIAR
del tracking_fragment*/
private fun startTimer(){
    /*Dentro de la función que realiza el seguimiento de ruta en primer plano por medio de la
    notificación se utiliza la función que añade la ruta pintada vacía inicial, esto se debe
    hacer desde la función que inicializa el temporizador porque ahí es donde se deberá dejar
    de pintar la ruta en el mapa, ya sea porque se ha pausado el servicio de seguimiento o
    porque es la primera vez que se ha corrido*/
    addEmptyPolyline()
    /*Comos se ha iniciado el seguimiento de la ruta, se debe poner como true el valor de la
    variable isTracking que indica si se está realizando el monitoreo de ruta o no y se empieza
    a guardar el tiempo inicial de la ruta en milisegundos dentro de la variable timeStarted e
    indicar por medio de la variable isTimmerEnabled que el monitoreo de tiempo ha inciado*/
    isTracking.postValue(true)
    timeStarted = System.currentTimeMillis()
    isTimmerEnabled = true
    /*Para monitorear el tiempo del recorrido se utilizará la librería de Corrutinas, porque no
    se quiere llamar a los objetos observadores siempre, solamente se hará esporádicamente para
    que la app no se vaya a trabar y por medio de corrutinas se hará en segundo plano y se
    ejecutará en paralelo con otras acciones del programa, esto se realizará con un bucle while
    que se ejecutará mientras se esté realizando el monitoreo del tiempo*/
    CoroutineScope(Dispatchers.Main).launch {
        /*Mientras el valor de la variable isTracking sea true, osea que se esté realizando el
        monitoreo del tiempo, se ejecutará el bucle while para registrar el tiempo de la ruta
        tomando en cuenta el tiempo inicial*/
        while (isTracking.value!!){
            //Diferencia de tiempo entre el tiempo inicial y el tiempo actual
            lapTime = System.currentTimeMillis() - timeStarted
            //Actualización de la variable que instancía la clase MutableLiveData
            timeRunInMillis.postValue(timeRun + lapTime)
            /*Ahora se creará el condicional que verá si el último segundo que ha pasado en
            milisegundos es mayor o menos al último milisegundo transcurrido + otros 1,000
            milisegundos, esto se hace para transformar el tiempo de milisegundos a segundos*/
            if(timeRunInMillis.value!! >= lastSecondTimeStamp + 1000L){
                timeRunInSeconds.postValue(timeRunInSeconds.value!! + 1)
                lastSecondTimeStamp += 1000L
                /*Ahora se debe retrasar la corrutina para que el observador no se esté
                actualizando siempre, sino que lo haga cada 50 miliegundos, este tiempo del
                delay estará guardado dentro del archivo de Constants*/
            }
            delay(TIMER_UPDATE_INTERVAL)
        }
        /*Fuera del bucle while se usará la variable timeRun que almacena el tiempo total del
        recorrido y se le añadirá el valor de la variable lapTime que almacena el tiempo del
        recorrido actual para obtener el tiempo total del recorrido*/
        timeRun += lapTime
    }
}

/*Función que pausa el seguimiento de la ruta*/
private fun pauseService(){
    isTracking.postValue(false)
    /*Cuando se pausa el recorrido, también se pausa el monitoreo del tiempo por lo que a la
    variable isTimerEnabled que hace ese monitoreo se le da un valor booleano de false para que
    deje de monitorear el tiempo*/
```

```
            isTimmerEnabled = false
        }

        /*Función que actualiza el estado de las notificaciones, recibe como parámetro la variable
        isTracking que indica si el recorridoha sido inicializado por medio del botón de INICIAR o no*/
        private fun updateNotificationTrackingState(isTracking: Boolean){
            /*El texto de la notificación dependerá del estado del recorrido, que puede ser PAUSAR o
            REANUDAR, si se está realizando el recorrido en la notificación deberá decir PAUSAR y si
            el recorrido está pausado debe decir REANUDAR*/
            val notificationActionText = if(isTracking) "PAUSAR" else "REANUDAR"
            /*El comando (intent) pendiente debe estar relacionado a los comandos para que se pueda
            indicar al servicio de Google Play que debe pasar si se da clic en el comando (action)*/
            val pendingIntent = if(isTracking){
                /*Comando mandado al servicio para indicar que debe pausarse por medio de la clase
                TrackingService*/
                val pauseIntent = Intent(this, TrackingService::class.java).apply {
                    //Constante para el comando de PAUSAR que se encuentra en el archivo Constants
                    action = ACTION_PAUSE_SERVICE
                }
                PendingIntent.getService(this, 1, pauseIntent, FLAG_UPDATE_CURRENT)
                /*Si el servicio de rastreo no ha sido inicializado la acción será REANUDAR el rastreo*/
            }else{
                val resumeIntent = Intent(this, TrackingService::class.java).apply {
                    action = ACTION_START_OR_RESUME_SERVICE
                }
                PendingIntent.getService(this, 2, resumeIntent, FLAG_UPDATE_CURRENT)
            }
            /*Referencia para el gestor de la notificación que se convierte en un objeto
            NotificationManager para poder actualizar la ventana de notificaciones, con esto se crea
            una lista vacía de ventanas de notificación que se irán llenando con datos conforme se
            vayan actualizando los milisegundos del temporizador*/
            val notificationManager = getSystemService(Context.NOTIFICATION_SERVICE) as
NotificationManager
            currentNotificationBuilder.javaClass.getDeclaredField("mActions").apply {
                isAccessible = true
                set(currentNotificationBuilder, ArrayList<NotificationCompat.Action>())
            }
            /*Antes de ejecutar esta parte de la función se debe checar si el servicio de rastreo ha
            sido detenido con la función killService*/
            if(!serviceKilled){
                /*Actualización de la ventana de notificación que contiene un ícono de pausa para pausar
                el seguimiento de la ruta*/
                currentNotificationBuilder = baseNotificationBuilder
                    .addAction(R.drawable.ic_pause_black_24dp, notificationActionText, pendingIntent)
                /*Actualización de la ventana de notificación lanzando distintas ventanas de
notificación con el mismo id para que parezca que es la misma, cada que se aumente en un segundo el
                tiempo del recorrido*/
                notificationManager.notify(NOTIFICATION_ID, currentNotificationBuilder.build())
            }
        }

        /*Se creará una última función que actualice el seguimiento de ruta usando como parámetro la
        variable isTracking de tipo boolean*/
        @SuppressLint("MissingPermission")
        private fun updateLocationTracking(isTracking: Boolean){
            /*Si la variable isTracking vale true, debemos checar si tenemos los permisos de rastreo de
            ubicación usando el método hasLocationPermissions del objeto TrackingUtility para que se
            empiece a rastrear la ruta*/
            if(isTracking){
                if(TrackingUtility.hasLocationPermissions(this)){
                    /*Si se tiene el permiso para el seguimiento de ruta, se debe solicitar
                    actualizaciones de localización*/
                    val request = LocationRequest().apply {
```

```
                /*El intervalo de muestreo para la ubicación se declaró en el archivo de
                Constants del paquete other y su unidad es de milisegundos, en específico se
                declaró que sea de 5,000 milisegundos para que no gaste tantos recursos del
                dispositivo móvil, además se debe indicar el tiempo de muestreo maximo, que fue
                declarado de 2,000 milisegundos y que la medición debe ser de alta precisión*/
                interval = LOCATION_UPDATE_INTERVAL
                fastestInterval = FASTEST_LOCATION_UPDATE_INTERVAL
                priority = PRIORITY_HIGH_ACCURACY
            }
            /*Se utiliza el objeto fusedlocationProviderClient para poder realizar las
            actualizaciones del seguimiento de la ruta que utiliza la función callback creada
            en este mismo archivo, el objeto request y un método llamado
            Looper.getMainLooper()*/
            fusedLocationProviderClient.requestLocationUpdates(
                request,
                locationCallback,
                Looper.getMainLooper()
            )
        }
    }else{
        /*Si el seguimiento de ruta ha sido parado, se debe remover las actualizaciones de
        localización hechas con el objeto fusedlocationProviderClient*/
        fusedLocationProviderClient.removeLocationUpdates(locationCallback)
    }
}

/*Ahora se debe definir un callback para la ubicación, ya que para obtener las actualizaciones
de posición es necesario usar una herramienta llamada Fuse Location Provider Client, que es una
parte de la API de Google Services que nos dará actualizaciones de ubicación en función de un
intervalo indicado en la herramienta*/
var locationCallback = object : LocationCallback(){
    override fun onLocationResult(result: LocationResult?) {
        super.onLocationResult(result)
        /*Esta línea de código checa si ya se está realizando el seguimiento de ruta*/
        /*Esto guardará el estado actual de la ubicación en la variable del parámetro result*/
        if(isTracking.value!!) {
            result?.locations?.let { locations ->
                /*Y se usará un bucle for para realizar el seguimiento de la ruta si es que
                se está haciendo el seguimiento, se llama la función addPathPoint() para que
                empiece a dibujar la ruta*/
                for(location in locations){
                    addPathPoint(location)
                    /*Para probar la funcionalidad del seguimiento de ruta se agrega dentro de
                    la función callback una impresión en consola*/
                    Timber.d("NUEVA UBICACIÓN: ${location.latitude}, ${location.longitude}")
                }
            }
        }
    }
}

/*Para el seguimiento de ruta también se necesita una función que añada una línea de coordenadas
a la última lista de lista de coordenadas (llamado en este archivo como Polyline) haciendo uso
de un objeto Location proveniente de la API de Google Maps, para checar si este es diferente a
null, osea que se ha iniciado la ruta*/
private fun addPathPoint(location: Location?){
    location?.let {
        /*Ahora se convierte el objeto Location en tipo LatLng que representan coordenadas de
        Latitud y Longitud*/
        val pos = LatLng(location.latitude, location.longitude)
        /*Posteriormente se añade la variable posición a la última posición de la lista de
        listas de coordenadas, que es de tipo Polyline y al seguir añadiendo valores, se pone
        el nuevo valor */
```

```
            pathPoints.value?.apply {
                last().add(pos)
                pathPoints.postValue(this)
            }
        }
    }

    /*La siguiente función hará que el tipo de dato Polyline esté vacío para cuando se ponga el
    botón de PARAR en la aplicación y luego se dé clic en el de REANUDAR, esa parte debe estar
    vacía*/
    private fun addEmptyPolyline() = pathPoints.value?.apply {
        /*Con esto se agrega una lista vacía*/
        add(mutableListOf())
        pathPoints.postValue(this)
    /*Si la ruta todavía no ha sido iniciada se pondrá una línea vacía inicial declarada en la
    función postInitialValues donde se indican los valores iniciales*/
    } ?: pathPoints.postValue(mutableListOf(mutableListOf()))

    /*Con esta función se inicializa el servicio en primer plano y se activa solamente cuando se ha
    iniciado el servicio de seguimiento de ruta de Google Play Services por primera vez, esto se
    usa para crear la notificación de la aplicación y que algunos datos se sigan mostrando en la
    parte superior del dispositivo aún cuando la aplicación haya sido cerrada*/
    private fun startForegroundService(){
        /*Se inicializa el monitoreo del tiempo del recorrido por medio de la función starTimer()*/
        startTimer()

        /*Se inicializa el valor de la variable isTracking como true porque se ha iniciado el
        monitoreo de la ruta por medio del servicio en primer plano*/
        isTracking.postValue(true)

        val notificationManager = getSystemService(Context.NOTIFICATION_SERVICE)
            as NotificationManager
        /*Después se debe checar si el sistema operativo del dispositivo es de versión Android Oreo
        o una superior para crear un canal de comuicación entre la Application de la aplicación y
        el sistema operativo*/
        if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.O){
            createNotificationChannel(notificationManager)
        }
        /*Aquí se crea la notificación, para ello debemos pasarle el ID del canal de comunicación
        creado como constante en el archivo de Constants que se encuentra en el paquete de other.
        Luego se le aplica el método .setAutoCancel(false) para prevenir que si el usuario da clic
        sin querer en la notificación, esta desaparezca.
        Se aplica el método .setOngoing(true) para que la notificación pueda ser removida
        con un movimiento de dedo sobre él.
        Después se aplica el método .setSmallIcon(R.drawable.ic_directions_run_black_24dp) para
        que se muesre con un ícono en específico la notificación que se encuentra en la carpeta
        app/res(o src)/drawable.
        El método .setContentTitle("Aplicación Enerdrais") para añadir el título de la ventana de
        notificación.
        Y el método .setContentText("00:00:00") para mostrar el texto que aparecerá en la
        notificación, que en este caso se usa para mostrar el tiempo de inicio del recorrido, que
        cada notificación actualizará el tiempo aumentándolo.
        Se dejó de usar cuando se creó este elemento por medio de inyección de dependencias con
        Dagger Hilt*/
        /*val notificationBuilder = NotificationCompat.Builder(this, NOTIFICATION_CHANNEL_ID)
            .setAutoCancel(false)
            .setOngoing(true)
            .setSmallIcon(R.drawable.ic_directions_run_black_24dp)
            .setContentTitle("Aplicación Enerdrais")
            .setContentText("00:00:00")
            .setContentIntent(getMainActivityPendingIntent())*/

        /*Ya con esto se puede lanzar el servicio en primer plano con la siguiente línea de código
```

```
        asignándole la id creada en el archivo de constants llamado NOTIFICATION_ID*/
        startForeground(NOTIFICATION_ID, baseNotificationBuilder.build())

        /*Después de inicializar el servicio en primer plano que muestra la ventana de
        notificaciones aún cuando se haya cerrado la aplicación se usa la varible timeRunInSeconds
        para crear un objeto observador y actualizar la ventana de notificaciones, el tiempo
        obtenido de la función getFormattedStopWatchTime obtenida de la clase TrackingUtility se
        debe multiplicar por 1000 ya que viene en milisegundos por default*/
        timeRunInSeconds.observe(this, Observer {
            /*Dentro de la función se comprueba que el servicio no haya sido detenido con la función
            killService, osea cuando la variable serviceKilled es igual a true, donde su valor
            inicial es false y la ventana de notificaciones desaparecerá*/
            if(!serviceKilled){
                /*Actualización de la ventana de navegación*/
                val notification = currentNotificationBuilder
                    .setContentText(TrackingUtility.getFormattedStopWatchTime(it*1000))
                notificationManager.notify(NOTIFICATION_ID, notification.build())
            }
        })
    }

    /*La función anterior crea comandos (intents) pendientes, para que estos puedan ser lanzados
    se debe usar la siguiente función que utiliza el método PendingIntent y una acción guardara en
    el archivo de Contants llamada ACTION_SHOW_TRACKING_FRAGMENT con una bandera de actualización
    de notificaciones pendientes, la cual actualiza el registro de notificaciones pendientes y las
    va lanzando una por una.
    Se dejó de usar cuando se creó este elemento por medio de inyección de dependencias con
    Dagger Hilt*/
    /*private fun getMainActivityPendingIntent() = PendingIntent.getActivity(
        this,
        0,
        Intent(this, MainActivity::class.java). also{
            it.action = ACTION_SHOW_TRACKING_FRAGMENT
        },
        FLAG_UPDATE_CURRENT
    )*/

    /*Este será un servicio ejecutado en primer plano para que no pueda ser detenido por el sistema
    operativo Android, perdiendo todos los datos del recorrido y que además el usuario siempre esté
    consciente de que se está corriendo, para ello se debe crear una ventana de notificación al
    usuario donde se le indique que el servicio va a empezar a correr, la importancia del método
    será baja ya que cada segundo que pase se enviará una notificación para actualizar los datos del
    recorrido. El método NotificationChannel necesita ser creado usando la anotación Requires API y
    usando el sistema operativo Oreo, por medio de la línea de código:
    @RequiresApi(Build.VERSION_CODES.O)*/
    @RequiresApi(Build.VERSION_CODES.O)
    private fun createNotificationChannel(notificationManager: NotificationManager){
        val channel = NotificationChannel(
            NOTIFICATION_CHANNEL_ID,
            NOTIFICATION_CHANNEL_NAME,
            IMPORTANCE_LOW
        )
        notificationManager.createNotificationChannel(channel)
    }

}
```

CanceltrackingDialog

```
package com.androiddevs.runningappyt.ui.fragments

import android.app.Dialog
```

```
import android.os.Bundle
import androidx.fragment.app.DialogFragment
import com.androiddevs.runningappyt.R
import com.google.android.material.dialog.InsetDialogOnTouchListener
import com.google.android.material.dialog.MaterialAlertDialogBuilder

class CancelTrackingDialog : DialogFragment() {

    private var yesListener : (() -> Unit)? = null

    fun setYesListener(listener: () -> Unit){
        yesListener = listener
    }

    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
        return MaterialAlertDialogBuilder(requireContext(), R.style.AlertDialogTheme)
            .setTitle("¿CANCELAR EL RECORRIDO?")
            .setMessage("¿Seguro deseas cancelar el recorrido actual y borrar los datos de su
rendimiento?")
            .setIcon(R.drawable.ic_delete)
            .setPositiveButton("Sí"){_, _ ->
                /*Lo que pasará cuando se dé clic al botón de Sí quiero cancelar mi recorrido es
                definido por esta función lambda a la que le pasamos parámetros vacíos y dentro
                se usa la función stopRun() del archivo TrackingFragment*/
                yesListener?.let { yes ->
                    yes()
                }
            }
            .setNegativeButton("No"){dialogInterface, _ ->
                /*Lo que pasará cuando se dé clic al botón de No quiero cancelar mi recorrido es
                que no se mostrará nada en pantalla pero no se cancelará el recorrido*/
                dialogInterface.cancel()
            }
            /*See crea y muestra la ventana emergente de confirmación que aparece cuando se quiere
            cancelar un recorrido y no guardar sus datos de rendimiento en la base de datos Room
            de la aplicación*/
            .create()
    }
}
```

RunFragment

```
package com.androiddevs.runningappyt.ui.fragments
import android.Manifest
import android.os.Build
import android.os.Bundle
import android.view.View
import android.widget.AdapterView
import androidx.appcompat.widget.LinearLayoutCompat
import androidx.fragment.app.Fragment
import androidx.fragment.app.viewModels
import androidx.lifecycle.LifecycleOwner
import androidx.lifecycle.Observer
import androidx.navigation.fragment.findNavController
import androidx.recyclerview.widget.LinearLayoutManager
import com.androiddevs.runningappyt.R
import com.androiddevs.runningappyt.adapters.RunAdapter
import com.androiddevs.runningappyt.other.Constants.REQUEST_CODE_LOCATION_PERMISSION
import com.androiddevs.runningappyt.other.SortType
import com.androiddevs.runningappyt.other.TrackingUtility
import com.androiddevs.runningappyt.ui.viewmodels.MainViewModel
import dagger.hilt.android.AndroidEntryPoint
```

```
import kotlinx.android.synthetic.main.fragment_run.*
import pub.devrel.easypermissions.AppSettingsDialog
import pub.devrel.easypermissions.EasyPermissions

/*La clase RunFragment hereda de la clase Fragment. Un fragmento representa  una parte reutilizable
de la interfaz de la app, define y administra su propio diseño, que puede ser reutilizado para
mostrar distintos datos de forma dinámica, tiene su propio ciclo de vida y puede administrar sus
propios eventos de entrada. Los fragmentos no pueden existir por sí solos, sino que deben estar
alojados por una Actividad (página cualquiera de la aplicación Android) u otro fragmento.*/

/*La clase R se refiere a resoruces que significa recursos (rsc) y sirve para acceder a las demás
carpetas del proyecto para utilizar distintos elementos como imágenes, colores, etc. En este caso
se utiliza para jalar layouts, que es donde se encuentran los elementos de estilo de la aplicación
dentro de la carpeta app/res/layout*/

/*Para usar la librería Dagger Hilt es necesario inyectar componentes a través de la anotación
@AndroidEntryPoint*/
@AndroidEntryPoint
/*Para el permiso del rastreo de localización se debe implementar la interfaz
EasyPermissions.PermissionCallbacks proveniente de la librería easy permissions*/
class RunFragment : Fragment(R.layout.fragment_run), EasyPermissions.PermissionCallbacks{
    /*Para inyectar el viewmodel que ya tiene incluido el acceso al repositorio es necesario crear
    un objeto MainViewModel y uilizar el método viewModels() para crear una fábrica de ViewModels
    por medio del uso de la librería de Dagger Hilt*/
    /*Se diseñó la clase ViewModel con el fin de almacenar y administrar datos relacionados con la
    interfaz de manera optimizada para los ciclos de vida. La clase ViewModel permite que se
    conserven los datos luego de cambios de configuración, como las rotaciones de pantallas.*/

    private val viewModel : MainViewModel by viewModels()
    /*Se coloca el RecyclerView creado para mostrar las rutas completadas al usuario por medio de
    una variable que sea una instancia de la clase RunAdapter donde se creó el RecyclerView*/
    private lateinit var runAdapter: RunAdapter

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        /*Se llama la función requestPermissions() creada debajo para poder pedir los permisos
        cuando se entre al fragmento fragment_run*/
        requestPermissions()

        /*Función que crea los RecyclerView en el fragmento fragment_run que observará cambios en
        la base de datos jalada de un observador creado en el archivo MainViewModel llamado
        runs que recopila los datos de la base de datos a través del repositorio de la app, los
        recorridos serán acomodados del mayor al menor en los valores seleccionados*/
        setupRecyclerView()
        /*Dependiendo del tipo de dato mostrado en un ViewModel se acomodarán de cierta manera
        los datos para que se coloquen de forma organizada en el fragmento item_run en función del
        elemento elegido de un array llamado filter_options que se encuentra en el archivo
        app/java/res/values/strings.xml, en la primera posición se elige que se acomoden los
        elementos por:
        0.- Fecha
        1.- Tiempo del recorrido
        2.- Distancia
        3.- Velocidad promedio
        4.- Calorías quemadas*/
        when(viewModel.sortType){
            SortType.DATE -> spFilter.setSelection(0)
            SortType.RUNNING_TIME -> spFilter.setSelection(1)
            SortType.DISTANCE -> spFilter.setSelection(2)
            SortType.AVG_SPEED -> spFilter.setSelection(3)
            SortType.CALORIES_BURNED -> spFilter.setSelection(4)
        }
        /*Función para responder a los cambios del elemento que acomoda los datos por si se quiere
        cambiar la forma de ordenar los datos de los recorridos*/
```

```
        spFilter.onItemSelectedListener = object : AdapterView.OnItemSelectedListener{
            /*La función onItemSelected sirve para indicar el viewModel que se está usando, la
            posición donde se coloca y su id asignado*/
            override fun onItemSelected(adapterView: AdapterView<*>?, view: View?, pos: Int, id:
Long) {
                /*Dependiendo de la posición del ícono donde se dé clic se acomodarán de cierta
                manera los datos siguiendo las mismas coordenadas descritas en el array
                filter_options que se encuentra en el archivo app/java/res/values/strings.xml*/
                when(pos){
                    0 -> viewModel.sortRuns(SortType.DATE)
                    1 -> viewModel.sortRuns(SortType.RUNNING_TIME)
                    2 -> viewModel.sortRuns(SortType.DISTANCE)
                    3 -> viewModel.sortRuns(SortType.AVG_SPEED)
                    4 -> viewModel.sortRuns(SortType.CALORIES_BURNED)
                }
            }
            //La función onNothingSelected no es necesario que la usemos pero si debía ser incluida
            override fun onNothingSelected(p0: AdapterView<*>?) { }
        }
        viewModel.runs.observe(viewLifecycleOwner, Observer {
            runAdapter.submitList(it)
        })

        /*Utilizando la clase viewModels se hace uso de la librería Dagger para que maneje la
inyección
        de dependencias creando una fábrica de viewModels en segundo plano*/
        fab.setOnClickListener {
            findNavController().navigate(R.id.action_runFragment_to_trackingFragment)
        }
    }

    /*Función que cree los RecyclerView para colocarlo en el item rvRuns del fragmento
    fragment_run*/
    private fun setupRecyclerView() = rvRuns.apply {
        runAdapter = RunAdapter()
        adapter = runAdapter
        layoutManager = LinearLayoutManager(requireContext())
    }

    /*Función para solicitar los permisos de ubicación que utiliza la función declarada en el
    objeto Tracking Utility del paquete other*/
    private fun requestPermissions(){
        if(TrackingUtility.hasLocationPermissions(requireContext())){
            /*Si ya se cuenta con el permiso de ubicación se termina la función*/
            return
        }
        /*Si el usuario no ha concedido el acceso al rastreo de ubicación se checará si el sistema
        operativo del usuario está corriendo en Android Q o Android normal para ejecutar alguna de
        las funciones del objeto TrackingUtility*/
        if(Build.VERSION.SDK_INT < Build.VERSION_CODES.Q){
            EasyPermissions.requestPermissions(
                /*A la función se le debe pasar como parámetros el Fragmento con el que debe
                trabajar y en este caso eso se hace con el operador this, que se refiere al mismo
                elemento donde se encuentra la aplicación en esta clase, que está relacionada con
                el fragmento deseado, como segundo parámetro solicita un mensaje que se debe
                imprimir en pantalla para ser mostrado al usuario y por último se necesita poner
                una constante que indique el permiso necesario para ejecutar la función que se
                encuentra en el objeto TrackingUtility que utiliza la librería EasyPermissions.
                La constante utilizada para solicitar el permiso se encuentra en el objeto
                Constants del paquete other*/
                this,
                "Necesitas aceptar el permiso de ubicación para empezar tu recorrido con la
aplicación",
```

```
                REQUEST_CODE_LOCATION_PERMISSION,
                /*Como no se encuentra en un dispositivo con sistema operativo Android Q, solo es
                necesario solicitar los permisos ACCESS_COARSE_LOCATION y ACCESS_FINE_LOCATION*/
                Manifest.permission.ACCESS_COARSE_LOCATION,
                Manifest.permission.ACCESS_FINE_LOCATION
            )
        }else{
            /*Si el sistema operativo del dispositivo móvil es Android Q se realiza lo mismo que en
            el contenido del if con la diferencia que también se debe solicitar el permiso de
            ACCESS_BACKGROUND_LOCATION*/
            EasyPermissions.requestPermissions(
                this,
                "Necesitas aceptar el permiso de ubicación para empezar tu recorrido con la
aplicación",
                REQUEST_CODE_LOCATION_PERMISSION,
                Manifest.permission.ACCESS_COARSE_LOCATION,
                Manifest.permission.ACCESS_FINE_LOCATION,
                Manifest.permission.ACCESS_BACKGROUND_LOCATION
            )
        }
    }

    /*Se debe implementar una función para manejar la interfaz EasyPermissions.PermissionCallbacks
    proveniente de la librería easy permissions*/
    /*Esta función checa si el usuario permanentemente denegó los permisos de ubicación para
    mostrar el diálogo utilizado en la función anterior y sino se mostrará el siguiente mensaje
    redactado dentro de la función onPermissionsDenied que es una de las dos que hace funcionar la
    interfaz EasyPermissions.PermissionCallbacks*/
    override fun onPermissionsDenied(requestCode: Int, perms: MutableList<String>) {
        if(EasyPermissions.somePermissionPermanentlyDenied(this, perms)){
            /*Si el usuario denegó permanentemente los permisos se le dirigirá al usuario a las
            configuraciones del teléfono para que pueda habilitar los permisos*/
            AppSettingsDialog.Builder(this).build().show()
        }else{
            /*Si denegó los permisos por primera vez o los denegó de forma no permanente se le
            solicitan otra vez los permisos*/
            requestPermissions()
        }
    }
    /*Si el usuario concedió el permiso de ubicación no se le mostrará nada en pantalla, pero es
    necesario dejar esta función en el código para satisfacer la interfaz
    EasyPermissions.PermissionCallbacks*/
    override fun onPermissionsGranted(requestCode: Int, perms: MutableList<String>) {}

    /*La siguiente función maneja el resultado del permiso introducido a la app, dentro de esta es
    donde se utilizan las funciones onPermissionsDenied y onPermissionsGranted que hacen funcionar
    la interfaz*/
    override fun onRequestPermissionsResult(
        requestCode: Int,
        permissions: Array<out String>,
        grantResults: IntArray
    ) {
        /*Llama la iterfaz para ver el resultado de los permisos a la librería easy permissions*/
        super.onRequestPermissionsResult(requestCode, permissions, grantResults)
        EasyPermissions.onRequestPermissionsResult(requestCode, permissions, grantResults, this)
    }
}
```

SettingsFragment

```
package com.androiddevs.runningappyt.ui.fragments
import android.content.SharedPreferences
```

```
import android.os.Bundle
import android.view.View
import androidx.fragment.app.Fragment
import com.androiddevs.runningappyt.R
import com.androiddevs.runningappyt.other.Constants.KEY_NAME
import com.androiddevs.runningappyt.other.Constants.KEY_WEIGHT
import com.google.android.material.snackbar.Snackbar
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.android.synthetic.main.activity_main.*
import kotlinx.android.synthetic.main.fragment_settings.*
import javax.inject.Inject

/*La clase RunFragment hereda de la clase Fragment. Un fragmento representa  una parte reutilizable
de la interfaz de la app, define y administra su propio diseño, que puede ser reutilizado para
mostrar distintos datos de forma dinámica, tiene su propio ciclo de vida y puede administrar sus
propios eventos de entrada. Los fragmentos no pueden existir por sí solos, sino que deben estar
alojados por una Actividad (página cualquiera de la aplicación Android) u otro fragmento.*/

/*La clase R se refiere a resoruces que significa recursos (rsc) y sirve para acceder a las demás
carpetas del proyecto para utilizar distintos elementos como imágenes, colores, etc. En este caso
se utiliza para jalar layouts, que es donde se encuentran los elementos de estilo de la aplicación
dentro de la carpeta app/res/layout*/

/*Para usar la librería Dagger Hilt es necesario inyectar componentes a través de la anotación
@AndroidEntryPoint*/
@AndroidEntryPoint
class SettingsFragment : Fragment(R.layout.fragment_settings){
    /*Para guardar los cambios en la shared preference de la base de datos se crea un late init var
    que sea una instancia de la clase SharedPreferences en conjunto con la anotación @Inject para
    que la librería Dagger Hilt inyecte las dependencias necesarias para que desde este archivo se
    pueda acceder a la base de datos, en conjunto con la anotación @AndroidEntryPoint*/
    @Inject
    lateinit var sharedPreferences : SharedPreferences

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        loadFieldsFromSharedPref()
        btnApplyChanges.setOnClickListener {
            val success = applyChangesToSharedPref()
            if(success){
                Snackbar.make(view, "Hemos actualizado sus datos", Snackbar.LENGTH_LONG).show()
            }else{
                Snackbar.make(view, "Por favor ingresa todos tus datos",
Snackbar.LENGTH_LONG).show()
            }
        }
    }

    /*Esta función sirve para cargar los cambios a la base de datos*/
    private fun loadFieldsFromSharedPref(){
        val name = sharedPreferences.getString(KEY_NAME, "")
        val weight = sharedPreferences.getFloat(KEY_WEIGHT, 80f)
        etName.setText(name)
        etWeight.setText(weight.toString())
    }

    /*En este fragmento se permite cambiar el nombre o peso introducido al inciar por primera
    vez en la aplicación, con la función applyChangesToSharedPref() se realiza eso*/
    private fun applyChangesToSharedPref():Boolean{
        /*En esta variable se almacena el valor de lo que proviene de la propiedad text del elemento
        etName contenida en el fragmento fragment_settings, que tiene el mismo nombre que un
        elemento del fragmento fragment_setup*/
        val nameText = etName.text.toString()
```

```
        val weightText = etWeight.text.toString()
        /*Si alguno de estos parámetros se encuentra vacío*/
        if(nameText.isEmpty() || weightText.isEmpty()){
            return false
        }
        /*Para mandar los datos del settingsFragment hacia la base de datos en la parte de shared
        preferences se usa el método edit por medio de un objeto SharedPreferences por y los métodos
        de envío de datos, se ponen como parámetros del método la clave creada en el archivo
        Constants y la variable que contiene el valor a mandar, que es el nombre, el peso y cuando
        se hayan mandado correctamente los datos se cambiará el valor de la variable que monitorea
        si el usuario había entrado alguna vez a false. Se usa el método apply() para mandar los
        datos de una forma asíncrona*/
        sharedPreferences.edit()
            .putString(KEY_NAME, nameText)
            .putFloat(KEY_WEIGHT, weightText.toFloat())
            .apply()
        /*Como también se quiere cambiar el texto superior donde dice nuestro nombre ya habiendo
        mandado los datos actualizados a la base de datos se imprimirá en pantalla un mensaje que
        contiene el nombre del usuario en la propiedad text del elemento con id tvToolbarTitle que
        pertenece al fragmento activity_main*/
        val toolbarText = "!Vamos $nameText tu puedes!"
        requireActivity().tvToolbarTitle.text = toolbarText
        return true
    }
}
```

SetupFragment

```
package com.androiddevs.runningappyt.ui.fragments
import android.content.SharedPreferences
import android.os.Bundle
import android.view.View
import androidx.fragment.app.Fragment
import androidx.navigation.NavOptions
import androidx.navigation.fragment.findNavController
import com.androiddevs.runningappyt.R
import com.androiddevs.runningappyt.other.Constants.KEY_FIRST_TIME_TOGGLE
import com.androiddevs.runningappyt.other.Constants.KEY_NAME
import com.androiddevs.runningappyt.other.Constants.KEY_WEIGHT
import com.google.android.material.snackbar.Snackbar
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.android.synthetic.main.activity_main.*
import kotlinx.android.synthetic.main.fragment_setup.*
import javax.inject.Inject

/*La clase RunFragment hereda de la clase Fragment. Un fragmento representa  una parte reutilizable
de la interfaz de la app, define y administra su propio diseño, que puede ser reutilizado para
mostrar distintos datos de forma dinámica, tiene su propio ciclo de vida y puede administrar sus
propios eventos de entrada. Los fragmentos no pueden existir por sí solos, sino que deben estar
alojados por una Actividad (página cualquiera de la aplicación Android) u otro fragmento.*/

/*La clase R se refiere a resoruces que significa recursos (rsc) y sirve para acceder a las demás
carpetas del proyecto para utilizar distintos elementos como imágenes, colores, etc. En este caso
se utiliza para jalar layouts, que es donde se encuentran los elementos de estilo de la aplicación
dentro de la carpeta app/res/layout*/

/*Para usar la librería Dagger Hilt es necesario inyectar componentes a través de la anotación
@AndroidEntryPoint, esto se hace para pasar los datos ingresador por el usuario a la función
provideSharedPreferences del archivo AppModule*/
@AndroidEntryPoint
class SetupFragment : Fragment(R.layout.fragment_setup){
```

```
    /*Se debe inyectar las shared preferences por medio de la anotación @Inject y una variable
    llamada sharedPref que es instancia de la clase SharedPreferences*/
    @Inject
    lateinit var sharedPref: SharedPreferences

    /*Se debe checar si es la primera vez que el usuario utiliza la app o si ya la ha usado
    anteriormente, para que si ya ha ingresado anteriormente se salte y vaya directamente hacia el
    RunFragment, saltándose el SetupFragment, como es una variable que viene de base de datos se
    utiliza la anotación @Inject para recopilar el dato, pero como el tipo de dato Boolean es de
    tipo primitivo se debe poner como @set:Inject*/
    @set:Inject
    var isFirstAppOpen = true


    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        /*Se checa si es la primera vez que el usuario ha ingresado a la app o no, si no es su
        primera vez se navegará directamente al Run Fragment*/
        if(!isFirstAppOpen){
            /*No es la primera vez que el usuario ingresa a la app, por lo que se quitará del camino
            con el método .setPopUpTo() al fragmento setupFragment y se dirigirá al usuario por
            medio del método findNavController().navigate()*/
            val navOptions = NavOptions.Builder()
                .setPopUpTo(R.id.setupFragment, true)
                .build()
            findNavController().navigate(
                R.id.action_setupFragment_to_runFragment,
                savedInstanceState,
                navOptions
            )
        }

        /*Utilizando la clase viewModels se hace uso de la librería Dagger para que maneje la
        inyección de dependencias creando una fábrica de viewModels en segundo plano, en este caso
        eso se aplica al botón de Continue del fragmento fragment_setup*/
        tvContinue.setOnClickListener {
            /*Dentro del listener que monitorea el estado del botón continue se agrega una variable
            llamada success que guarda el valor booleano de la función writePersonalDataToSharedPref
            teniendo un registro así de si el usuario ha introducido sus datos o no*/
            val success = writePersonalDataToSharedPref()
            /*Si fue exitoso al dar clic en el botón de Continue, seguirá el flujo de la aplicación
            y el usuario será trasladado al siguiente fragmento de la apicación, sino se le mostrará
            un Snackbar indicándole que debe introducir sus datos*/
            if(success){
                /*Esta función lo que hace es ejecutar el diagrama de transiciones del archivo
                nav_graph para que al dar clic en el botón de continue se me lleve al siguiente
                fragmento*/
                findNavController().navigate(R.id.action_setupFragment_to_runFragment)
            }else{
                Snackbar.make(requireView(), "Por favor introduce todos tus datos para poder
ingresar", Snackbar.LENGTH_SHORT).show()
            }
        }
    }

    /*Función para indicar si el usuario ya ha ingresadon sus datos de nombre y peso o no, sino
    se utilizará una Snackbar para indcarle al usuario que debe introducir sus datos, esto se hace
    por medio de los id etName y etWeight que se encuentran en el fragment_setup para guardar ambos
    datos ingresados en una variable y convertirlos a String*/
    private fun writePersonalDataToSharedPref():Boolean{
        val name = etName.text.toString()
        val weight = etWeight.text.toString()
        /*Si el programa detecta que la variable name o weight se encuentran vacías se mostrará un
```

```
        mensaje avisando al usuario que debe ingresar sus datos por medio de una Snackbar*/
        if(name.isEmpty() || weight.isEmpty()){
            return false
        }
        /*Guardar los datos de name y weight en shared preference si es que las variables no están
        vacías, para mandar los datos del setupFragment hacia la base de datos en la parte de shared
        preferences se usa el método edit por medio de un objeto SharedPreferences por medio de los
        métodos de envío de datos, se ponen como parámetros del método la clave creada en el archivo
        Constants y la variable que contiene el valor a mandar, que es el nombre, el peso y cuando
        se hayan mandado correctamente los datos se cambiará el valor de la variable que monitorea
        si el usuario había entrado alguna vez a false. Se usa el método apply() para mandar los
        datos de una forma asíncrona*/
        sharedPref.edit()
            .putString(KEY_NAME, name)
            .putFloat(KEY_WEIGHT, weight.toFloat())
            .putBoolean(KEY_FIRST_TIME_TOGGLE, false)
            .apply()
        /*Ya habiendo mandado los datos a la base de datos se imprimirá en pantalla un mensaje que
        contiene el nombre del usuario en la propiedad text del elemento con id tvToolbarTitle que
        pertenece al fragmento activity_main*/
        val toolbarText = "!Vamos $name tu puedes!"
        requireActivity().tvToolbarTitle.text = toolbarText
        return true
    }
}
```

StatisticsFragment

```
package com.androiddevs.runningappyt.ui.fragments
import android.graphics.Color
import android.os.Bundle
import android.view.View
import androidx.core.content.ContextCompat
import androidx.fragment.app.Fragment
import androidx.fragment.app.viewModels
import androidx.lifecycle.Observer
import com.androiddevs.runningappyt.R
import com.androiddevs.runningappyt.other.CustomMarkerView
import com.androiddevs.runningappyt.other.TrackingUtility
import com.androiddevs.runningappyt.ui.viewmodels.MainViewModel
import com.androiddevs.runningappyt.ui.viewmodels.StatisticsViewModel
import com.github.mikephil.charting.components.XAxis
import com.github.mikephil.charting.data.BarData
import com.github.mikephil.charting.data.BarDataSet
import com.github.mikephil.charting.data.BarEntry
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.android.synthetic.main.fragment_statistics.*
import kotlinx.android.synthetic.main.item_run.*
import kotlin.math.round

/*La clase RunFragment hereda de la clase Fragment. Un fragmento representa  una parte reutilizable
de la interfaz de la app, define y administra su propio diseño, que puede ser reutilizado para
mostrar distintos datos de forma dinámica, tiene su propio ciclo de vida y puede administrar sus
propios eventos de entrada. Los fragmentos no pueden existir por sí solos, sino que deben estar
alojados por una Actividad (página cualquiera de la aplicación Android) u otro fragmento.*/

/*La clase R se refiere a resoruces que significa recursos (rsc) y sirve para acceder a las demás
carpetas del proyecto para utilizar distintos elementos como imágenes, colores, etc. En este caso
se utiliza para jalar layouts, que es donde se encuentran los elementos de estilo de la aplicación
dentro de la carpeta app/res/layout*/

/*Para usar la librería Dagger Hilt es necesario inyectar componentes a través de la anotación
```

```
@AndroidEntryPoint*/
@AndroidEntryPoint
class StatisticsFragment : Fragment(R.layout.fragment_statistics){
    /*Para inyectar el viewmodel que ya tiene incluido el acceso al repositorio es necesario crear
    un objeto StatisticsViewModel y uilizar el método viewModels() para crear una fábrica de
    ViewModels por medio del uso de la librería de Dagger Hilt*/
    /*Se diseñó la clase ViewModel con el fin de almacenar y administrar datos relacionados con la
    interfaz de manera optimizada para los ciclos de vida. La clase ViewModel permite que se
    conserven los datos luego de cambios de configuración, como las rotaciones de pantallas.*/
    /*Utilizando la clase viewModels se hace uso de la librería Dagger para que maneje la inyección
    de dependencias creando una fábrica de viewModels en segundo plano*/
    private val viewModel : StatisticsViewModel by viewModels()

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        subscribeToObservers()
        setupBarChart()
    }

    /*Función para modificar aspectos estéticos del gráfico de barras usado para mostrar datos de la
    ruta*/
    private fun setupBarChart(){
        barChart.xAxis.apply {
            position = XAxis.XAxisPosition.BOTTOM
            setDrawLabels(false)
            axisLineColor = Color.WHITE
            textColor = Color.WHITE
            setDrawGridLines(false)
        }
        barChart.axisLeft.apply {
            axisLineColor = Color.WHITE
            textColor = Color.WHITE
            setDrawGridLines(false)
        }
        barChart.axisRight.apply {
            axisLineColor = Color.WHITE
            textColor = Color.WHITE
            setDrawGridLines(false)
        }
        barChart.apply {
            description.text = "Velocidad promedio a través del tiempo"
            legend.isEnabled = false
        }
    }

    /*Función para imprimir en pantalla los elementos traídos de la base de datos sobre el
recorrido*/
    private fun subscribeToObservers(){
        viewModel.totalTimeRun.observe(viewLifecycleOwner, Observer {
            it?.let {
                val totalTimeRun = TrackingUtility.getFormattedStopWatchTime(it)
                tvTotalTime.text = totalTimeRun
            }
        })
        viewModel.totalDistance.observe(viewLifecycleOwner, Observer {
            it?.let {
                val km = it/1000f
                val totalDistance = round(km*10f)/10f
                val totalDistanceString = "${totalDistance} km"
                tvTotalDistance.text = totalDistanceString
            }
        })
        viewModel.totalAvgSpeed.observe(viewLifecycleOwner, Observer {
```

```
            it?.let {
                val avgSpeed = round(it*10f) /10f
                val avgSpeedString = "${avgSpeed} km/h"
                tvAverageSpeed.text = avgSpeedString
            }
        })
        viewModel.totalCaloriesBurned.observe(viewLifecycleOwner, Observer {
            it?.let {
                val totalCalories = "${it} kcal"
                tvTotalCalories.text = totalCalories
            }
        })
        /*Función para crear una gráfica de la velocidad promedio*/
        viewModel.runsSortedByDate.observe(viewLifecycleOwner, Observer {
            it?.let {
                val allAvgSpeeds = it.indices.map { i -> BarEntry(i.toFloat(), it[i].avgSpeedInKNMH)
}
                val barDataSet = BarDataSet(allAvgSpeeds, "Velocidad promedio a través del
tiempo").apply {
                    valueTextColor = Color.WHITE
                    color = ContextCompat.getColor(requireContext(), R.color.colorAccent)
                }
                barChart.data = BarData(barDataSet)
                barChart.marker = CustomMarkerView(it.reversed(), requireContext(),
R.layout.marker_view)
                barChart.invalidate()
            }
        })
    }
}
```

TrackingFragment.kt

```
package com.androiddevs.runningappyt.ui.fragments
import android.content.Intent
import android.os.Bundle
import android.view.*
import androidx.core.view.get
import androidx.fragment.app.Fragment
import androidx.fragment.app.viewModels
import androidx.lifecycle.Observer
import androidx.navigation.fragment.findNavController
import com.androiddevs.runningappyt.R
import com.androiddevs.runningappyt.db.Run
import com.androiddevs.runningappyt.other.Constants.ACTION_PAUSE_SERVICE
import com.androiddevs.runningappyt.other.Constants.ACTION_START_OR_RESUME_SERVICE
import com.androiddevs.runningappyt.other.Constants.ACTION_STOP_SERVICE
import com.androiddevs.runningappyt.other.Constants.MAP_ZOOM
import com.androiddevs.runningappyt.other.Constants.POLYLINE_COLOR
import com.androiddevs.runningappyt.other.Constants.POLYLINE_WIDTH
import com.androiddevs.runningappyt.other.TrackingUtility
import com.androiddevs.runningappyt.services.Polyline
import com.androiddevs.runningappyt.services.TrackingService
import com.androiddevs.runningappyt.ui.viewmodels.MainViewModel
import com.androiddevs.runningappyt.ui.viewmodels.StatisticsViewModel
import com.google.android.gms.maps.CameraUpdateFactory
import com.google.android.gms.maps.GoogleMap
import com.google.android.gms.maps.model.LatLng
import com.google.android.gms.maps.model.LatLngBounds
import com.google.android.gms.maps.model.PolygonOptions
import com.google.android.gms.maps.model.PolylineOptions
import com.google.android.material.dialog.MaterialAlertDialogBuilder
```

```
import com.google.android.material.snackbar.Snackbar
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.android.synthetic.main.fragment_tracking.*
import kotlinx.android.synthetic.main.fragment_tracking.view.*
import java.util.*
import javax.inject.Inject
import kotlin.math.round

/*Constante usada para que al realizar una rotación de pantalla todavía se pueda cancelar el
recorrido*/
const val CANCEL_TRACKING_DIALOG_TAG = "CancelDialog"

/*La clase RunFragment hereda de la clase Fragment. Un fragmento representa  una parte reutilizable
de la interfaz de la app, define y administra su propio diseño, que puede ser reutilizado para
mostrar distintos datos de forma dinámica, tiene su propio ciclo de vida y puede administrar sus
propios eventos de entrada. Los fragmentos no pueden existir por sí solos, sino que deben estar
alojados por una Actividad (página cualquiera de la aplicación Android) u otro fragmento.*/

/*La clase R se refiere a resoruces que significa recursos (rsc) y sirve para acceder a las demás
carpetas del proyecto para utilizar distintos elementos como imágenes, colores, etc. En este caso
se utiliza para jalar layouts, que es donde se encuentran los elementos de estilo de la aplicación
dentro de la carpeta app/res/layout*/

/*Para usar la librería Dagger Hilt es necesario inyectar componentes a través de la anotación
@AndroidEntryPoint*/
@AndroidEntryPoint
class TrackingFragment : Fragment(R.layout.fragment_tracking){
    /*Para inyectar el viewmodel que ya tiene incluido el acceso al repositorio es necesario crear
    un objeto StatisticsViewModel y uilizar el método viewModels() para crear una fábrica de
    ViewModels por medio del uso de la librería de Dagger Hilt*/
    private val viewModel : MainViewModel by viewModels()

    /*Variable global para poder dibujar la ruta en el mapa*/
    private var isTracking = false
    private var pathPoints = mutableListOf<Polyline>()

    /*Dentro del fragmento se utilizará el  MapView contenido dentro del fragmento del recorrido
    fragment_tracking para poder mostrar el mapa de Google maps dentro de este fragmento, pero para
    ello es necesario manejar su ciclo de vida ya que los MapView tienen su propio ciclo de vida
    que no es igual al del fragmento donde están contenidos, para ello se creará un objeto GoogleMap
    que es el objeto que contendrá al MapView que físicamente mostrará al mapa en pantalla*/
    /*Este objeto GoogleMap se utilizará para dibujar la ruta del usuario durante su recorrido sobre
    el MapView*/
    private var map: GoogleMap? = null

    /*Variable para mostrar el temporizador del recorrido que mostrará el tiempo total*/
    private var curTimeInMillis = 0L

    /*Variable donde se almacenará el peso del usuario introducido al iniciar la app, para ello se
    utiliza la anotación @set:Inject ya que float es un tipo de dato nativo y de esa forma se puede
    obtener el peso del usuario de la base de datos Room*/
    @set:Inject
    var weight = 80f

    /*Variable usada para manejar y crear el menú superior que tiene el botón de cancelar
recorrido*/
    private var menu: Menu? = null
    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        setHasOptionsMenu(true)
```

```
    return super.onCreateView(inflater, container, savedInstanceState)
}

/*Para inicializar el objeto GoogleMap se utiliza la siguiente función onViewCreated*/
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    /*Manejo del ciclo de vida: Donde se indica cuando se crea y cuando se destruye el mapView
    que muestra el mapa del seguimiento de ruta*/
    mapView.onCreate(savedInstanceState)
    /*Se asigna un listener al botón del fragmento fragment_tracking para que por medio de él
    se puedan mandar comandos (intents) al servicio de Google Play para manejar el mapa de
    Google Maps*/
    //EMPEZAR SERVICIO DE SEGUIMIENTO
    btnToggleRun.setOnClickListener {
        toggleRun()
    }
    if(savedInstanceState != null){
        val cancelTrackingDialog = parentFragmentManager.findFragmentByTag(
            CANCEL_TRACKING_DIALOG_TAG) as CancelTrackingDialog?
        cancelTrackingDialog?.setYesListener {
            stopRun()
        }
    }
    /*Botón para completar un recorrido, hacer zoom a la imagen del mapa con el recorrido
    dibujado, tomar una screen shot, convertirlo a Bitmap y guardar los datos en la base de
    datos Room*/
    btnFinishRun.setOnClickListener {
        zoomToSeeWholeTrack()
        endRunAndSaveToDb()
    }
    /*Dentro de usa el método getMapAsync para relacionar el objeto GoogleMap con el mapView
    que se encuentra dentro del fragmento fragment_tracking*/
    mapView.getMapAsync {
        map = it
        /*Función que manda a llamar todas las funciones que crean la linea de seguimiento de
        ruta*/
        addAllPolylines()
    }
    /*Función que manda a llamar todas las funciones que crean la linea de seguimiento de
    ruta*/
    subscribeToObservers()
}

/*Función que permite enviar y recibir los datos del recorrido en tiempo real hacia el servicio
de Google Play Services*/
private fun subscribeToObservers(){
    /*Observador de la clase TrackingService que usa la variable isTracking instanciada de la
    clase MutableList y recopila datos del entorno para monitorear si el servicio de
    seguimiento de ruta ha sido incializado o no*/
    TrackingService.isTracking.observe(viewLifecycleOwner, Observer {
        //Actualiza el estado de la función que monitorea el estado del servicio de rastre
        updateTracking(it)
    })
    /*Observador de la clase TrackingService que usa la variable pathPoints instanciada de la
    clase MutableList y recopila datos del entorno para crear la línea que dibuja la ruta del
    usuario en el mapa de Google Maps*/
    TrackingService.pathPoints.observe(viewLifecycleOwner, Observer{
        //Actualiza el estado de la variable que pinta la línea de la ruta sobre el mapa
        pathPoints = it
        //Ejecuta la función que actualiza la línea para que dibuje toda la ruta
        addLatestPolyLine()
        /*Ejecuta la función que actualiza el zoom sobre el mapa que se centra en la ruta del
        recorrido*/
```

```
            moveCameraToUser()
        })
        /*OObservador de la clase TrackingService que usa la variable pathPoints instanciada de la
        clase MutableList y recopila datos del entorno para actualizar el tiempo del recorrido
        mostrado en la aplicación*/
        TrackingService.timeRunInMillis.observe(viewLifecycleOwner, Observer {
            //Actualiza el estado de la variable que monitorea el temporizador del recorrido
            curTimeInMillis = it
            /*Actualiza el estado del temporizador usando la función getFormattedStopWatchTime() de
            la clae TrackingUtility, pasándole como parámetro la variable curTimeInMillis
            actualizada e incluye milisegundos en el temporizador*/
            val formattedTime = TrackingUtility.getFormattedStopWatchTime(curTimeInMillis, true)
            /*Se usa el id del elemento tvTimer dentro del fragmento fragment_tracking para mostrar
            el tiempo en pantalla*/
            tvTimer.text = formattedTime
        })
    }

    /*Función que muestra un botón para cambiar de estado de un recorrido ya iniciado, si está
    corriendo pararlo y si está parado reanudarlo usando los comandos creados para el seguimiento*/
    private fun toggleRun(){
        if(isTracking){
            /*Cuando el recorrido ha sido inicializado se muestra el botón de cancelar el
            recorrido en el elemento de tipo menú y en su primer item con el índice 0, que es el
            primero que haya, aunque en el menú superior solo existe un elemento*/
            menu?.getItem(0)?.isVisible = true
            /*Si el recorrido ya ha sido iniciado, se deberá pausar el recorrido*/
            sendCommandToService(ACTION_PAUSE_SERVICE)
        }else{
            /*Si el recorrido ya ha sido pausado, se deberá empezar o reanudar el recorrido*/
            sendCommandToService(ACTION_START_OR_RESUME_SERVICE)
        }
    }

    /*Utilización del archivo toolback_tracking_menu para crear el menú superior*/
    override fun onCreateOptionsMenu(menu: Menu, inflater: MenuInflater) {
        super.onCreateOptionsMenu(menu, inflater)
        inflater.inflate(R.menu.toolbar_tracking_menu, menu)
        this.menu = menu
    }

    /*Con esta función se puede cambiar la visibilidad del menú superior para mostrarlo solo en
    ciertas ocasiones, en específico se va a mostrar solo cuando se haya inicializado un recorrido*/
    override fun onPrepareOptionsMenu(menu: Menu) {
        super.onPrepareOptionsMenu(menu)
        if(curTimeInMillis > 0L){
            this.menu?.getItem(0)?.isVisible = true
        }
    }

    /*Esta función se utiliza para mostrar la ventana emergente de confirmación para cancelar un
    recorrido cuando se dé clic en el botón llamado miCancelTracking que se encuentra en el
    toolbar_tracking_menu, al hacerlo tambien aparecerá la ventana emergente creada con la función
    showCancelTrackingDialog()*/
    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        when(item.itemId){
            R.id.miCancelTracking ->{
                showCancelTrackingDialog()
            }
        }
        return super.onOptionsItemSelected(item)
    }
```

```
    /*Esta función crea una ventana de notificación para confirmar si el usuario en verdad quiere
    cancelar su recorrido por si acaso presionó el botón sin querer*/
    private fun showCancelTrackingDialog(){
        CancelTrackingDialog().apply {
            setYesListener {
                stopRun()
            }
        }.show(parentFragmentManager, CANCEL_TRACKING_DIALOG_TAG)
    }

    /*La función stopRun() sirve para detener el recorrido usando un comando que fue guardado en el
    archivo de Constants y regresar al fragmento de RunFragment donde se ven los recorridos del
    usuario*/
    private fun stopRun(){
        tvTimer.text = "00:00:00:00"
        sendCommandToService(ACTION_STOP_SERVICE)
        findNavController().navigate(R.id.action_trackingFragment_to_runFragment)
    }

    /*Función que observa los cambios de datos en el servicio de Google Play y reacciona ante
ellos*/
    private fun updateTracking(isTracking: Boolean){
        this.isTracking = isTracking
        /*Si no se está realizando el seguimiento de ruta el botón dirá INICIAR y estará visible
        * Si ya se está realizando el seguimiento de ruta el botón dirá DETENER y no se verá*/
        if(!isTracking && curTimeInMillis > 0L){
            btnToggleRun.text = "INICIAR"
            btnFinishRun.visibility = View.VISIBLE
        }else if(isTracking){
            btnToggleRun.text = "DETENER"
            /*Cuando el recorrido ha sido detenido se debe mostrar el botón de cancelar el
            recorrido por si el usuario quiere borrar todos sus datos de rendimiento*/
            menu?.getItem(0)?.isVisible = true
            btnFinishRun.visibility = View.GONE
        }
    }

    /*Función que mueve el zoom en el mapa para seguir la coordenada de la ruta*/
    private fun moveCameraToUser(){
        if(pathPoints.isNotEmpty() && pathPoints.last().isNotEmpty()){
            map?.animateCamera(
                CameraUpdateFactory.newLatLngZoom(
                    pathPoints.last().last(),
                    MAP_ZOOM
                )
            )
        }
    }

    /*Cuando el usuario dé clic en el botón de TERMINAR RECORRIDO se guardarán los parámetros de
    rendimiento en la base de datos de la app creada con la librería Room basada en SQLite que tal
    cual guarda los datos en el celular en vez de hacerlo en un servidor remotor. Para guardar los
    datos de rendimiento del recorrido en una base de datos es necesario guardarlos dependiendo de
    su tipo:
    Fecha del recorrido: Calendar (Clase de la que se crean objetos para dar fechas)
    Distancia recorrida en metros: Int (Entero)
    Velocidad promedio: Float (decimal)
    Imagen del mapa con la ruta dibujada: Tipo de dato Bitmap junto con las Polyline de las líneas
    dibujadas de la ruta.
    Para poder guardar la imagen de la ruta dibujada sobre el mapa con el mapa se utiliza una
    función que haga zoom sobre el mapa para que se vea toda la ruta y tome una foto para mostrarla
    al usuario como imagen*/
    private fun zoomToSeeWholeTrack(){
```

```
    /*Para este tipo de funcionalidad Google Maps tiene un objeto LatLng (latitud longitud)
    que dará al programa la parte del mapa en donde debe hacer zoom para capturar la ruta del
    recorrido a través de una variable donde estén guardadas todas las coordenadas del
    recorrido*/
    val bounds = LatLngBounds.Builder()
    /*Ya que se haya guardado en la variable las coordenadas del recorrido se usarán dos bucles
    for, uno para repasar las coordenadas completas de la línea dibujada sobre el mapa del
    recorrido que están guardadas en un tipo de dato polyline que es una lista de listas de
    coordenadas, por lo que uno de los bucles for recorerrá la lista de listas y el otro la
    lista de coordenadas para guardarlas todas en la variable bounds*/
    for(polyline in pathPoints){
        for(pos in polyline){
            bounds.include(pos)
        }
    }
    /*Ahora solo se moverá la cámara del mapa para hacer zoom en la zona de la ruta por medio
    del método moveCamera aplicado al objeto map instanciado de la clase de Google Maps para
    que se pueda tomar una foto y se le enseñe al usuario, para ello se le debe pasar la
    variable que contenga todas las coordenadas del recorrido, el ancho del mapa, la altura del
    mapa y el padding que se le quiere agregar para que tome una buena foto, para ello se
    elimina el 5% de la altura del mapa haciendo una multiplicación y el resultado se convierte
    a entero*/
    map?.moveCamera(
        CameraUpdateFactory.newLatLngBounds(
            bounds.build(),
            mapView.width,
            mapView.height,
            (mapView.height * 0.05f).toInt()
        )
    )
}

/*Función para terminar el recorrido y guardar los datos en una base de datos, para poder
ejecutar esta función es necesario que antes se haya corrido la función zoomToSeeWholeTrack()
para poder tomar una foto del mapa con la línea de la ruta y guardarla en la base de datos*/
private fun endRunAndSaveToDb(){
    /*Screen shot del mapa de Google maps con la ruta dibujada que entrega como resultado un
    dato de tipo Bitmap, para usar el método .snapshot que realiza esto es necesario conocer
    la distancia total del recorrido por medio de la función calculatePolyLineLength() que se
    encuentra en el archivo de TrackingUtility*/
    map?.snapshot { bmp->
        var distanceInMeters = 0
        for(polyline in pathPoints){
            distanceInMeters += TrackingUtility.calculatePolyLineLength(polyline).toInt()
        }
        /*Con la distancia total calculada del recorrido, dividirla entre 1000 para que la
        velocidad sea mostrada en km/h y el tiempo total del recorrido dividido también entre
        1000*60*60 para convertirlo a horas y así se puede calcular la velocidad promedio del
        usuario y mostrar solo un decimal, por lo que se usa el método round de la clase Math*/
        val avgSpeed = round(((distanceInMeters/1000f) /(curTimeInMillis/1000f/60/60))*10)/10f
        /*También se puede guardar la fecha y hora del recorrido usando el método
        Calendar.getInstance().timeInMillis*/
        val dateTimeStamp = Calendar.getInstance().timeInMillis
        /*Y se debe calcular las calorías quemadas usando la fórmula: d_km*peso*/
        val caloriesBurned = ((distanceInMeters/1000f)*weight).toInt()
        /*Objeto run instanciado de la clase Run del paquete db para pasar todos estos datos
        medidos a la base de datos al que se le pasa como parámetro:
        - El bitmap de la captura del mapa
        - La fecha
        - Velocidad promedio
        - Distancia en metros
        - Tiempo de duración
        - Calorías quemadas*/
```

```
            val run = Run(bmp, dateTimeStamp, avgSpeed, distanceInMeters, curTimeInMillis,
caloriesBurned)
            /*Se utiliza el método insertRun de la clase MainViewModel al cual se le pasa el objeto
            instanciado de la clase run recién creado para poder pasar los datos recopilados del
            recorrido*/
            viewModel.insertRun(run)
            /*Se usa la clase Snackbar para decir al usuario que el recorrido fue almacenado en la
            base de datos por medio del elemento rootView del fragment activity_main.xml*/
            Snackbar.make(
                requireActivity().findViewById(R.id.rootView),
                "Recorrido Enerdrais guardado con éxito!! Felicidades por completarlo",
                Snackbar.LENGTH_LONG
            ).show()
            /*Después de haber completado un recorrido se debe parar con la función stopRun()*/
            stopRun()
        }
    }

    /*Se debe crear la función que cree la línea de la ruta para cuando el dispositivo móvil sea
    rotado, así en cualquiera de las 2 opciones no se dejará de mostrar la línea de seguimiento de
    la ruta*/
    private fun addAllPolylines(){
        for(polyline in pathPoints){
            val polylineOptions = PolylineOptions()
                .color(POLYLINE_COLOR)
                .width(POLYLINE_WIDTH)
                .addAll(polyline)
            map?.addPolyline(polylineOptions)
        }
    }

    /*Función que dibuja la línea de seguimiento de ruta sobre el mapa, en específico conecta los
    últimos dos puntos de la lista de listas de coordenadas para crear la ruta*/
    private fun addLatestPolyLine(){
        /*Se debe checar si hay una lista de listas de coordenadas en el programa y si hay  más de
        un elemento para que se pueda empezar a crear la ruta*/
        if(pathPoints.isNotEmpty() && pathPoints.last().size > 1){
            /*En esta variable se guardará la penúltima coordenada de latitud y longitud*/
            val preLastLatLng = pathPoints.last()[pathPoints.last().size-2]
            /*Ahora solo tendremos que guardar en otra variable la última coordenada para poder
            unirlas*/
            val lastLatLng = pathPoints.last().last()

            /*Definición del color, el ancho de la línea de seguimiento de ruta, la conexión entre
            los dos últimos puntos para crear la línea y uso del objeto map que es instancia de la
            clase Google Maps para dibujar sobre el mapa la línea*/
            val polylineOptions = PolylineOptions()
                .color(POLYLINE_COLOR)
                .width(POLYLINE_WIDTH)
                .add(preLastLatLng)
                .add(lastLatLng)
            map?.addPolyline(polylineOptions)
        }
    }

    /*Función que manda al servidor los comandos (intents) al servicio de Google Maps por medio de
    la clase TrackingService que se encuentra dentro del paquete services donde se crearon los
    comandos de Inicializar y reanudar, pausar y parar el servicio de Google Maps*/
    private fun sendCommandToService(action: String) =
        Intent(requireContext(), TrackingService::class.java).also {
            it.action = action
            requireContext().startService(it)
        }
```

```
    /*Para manejar los ciclos del vida del MapView asociado al objeto GoogleMap se utilizan las
    funciones onResume, onStart, onStop, onPause, onLowMemory y onSaveInstanceState*/
    //Para cuando la aplicación fue pausada y luego se volvió a activar
    override fun onResume() {
        super.onResume()
        mapView?.onResume()
    }
    //Para cuando la aplicación fue iniciada
    override fun onStart() {
        super.onStart()
        mapView?.onStart()
    }
    //Para cuando la aplicación fue detenida
    override fun onStop() {
        super.onStop()
        mapView?.onStop()
    }
    //Para cuando la aplicación fue pausada
    override fun onPause() {
        super.onPause()
        mapView?.onPause()
    }
    /*Para cuando el dispositivo donde se encuentra la aplicación tiene baja batería se activa el
    modo ahorrador*/
    override fun onLowMemory() {
        super.onLowMemory()
        mapView?.onLowMemory()
    }
    /*Sirve para añadir el mapa por medio de memoria caché para que ya iniciada la aplicación no se
    deba volver a crear desde cero*/
    override fun onSaveInstanceState(outState: Bundle) {
        super.onSaveInstanceState(outState)
        mapView.onSaveInstanceState(outState)
    }
}
```

MainViewModel

```
package com.androiddevs.runningappyt.ui.viewmodels
import androidx.hilt.lifecycle.ViewModelInject
import androidx.lifecycle.MediatorLiveData
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.androiddevs.runningappyt.db.Run
import com.androiddevs.runningappyt.other.SortType
import com.androiddevs.runningappyt.repositories.MainRepository
import kotlinx.coroutines.launch

/*El propósito del MainViewModel es recolectar los datos recolectados del repositorio y proveerlos
para los fragmentos que los mostrarán en pantalla, los fragmentos son pedazos de la interfaz en la
aplicación Android que sirven para hacer que distintos datos dinámicos se puedan imprimir en una
misma zona de la aplicación*/

/*Para poder recolectar los datos del repositorio es necesario usar la anotación @ViewModelInject()
para crear una fábrica de ViewModels, esto se hace creando un objeto MainRepository que es instancia
de la clase MainRepository que es donde se creó el repositorio del proyecto que recolecta los datos
para pasárselos a la base de datos y a la interfaz de usuario Android, aunque no se puso
explícitamente un método para dar datos del repositorio, esto se puede hacer por medio de Dagger ya
que el parámetro de la función creada en MainRepository recibe como parámetro un objeto RunDAO de la
interfaz con el mismo nombre, esto fue descrito para la librería Dagger en el archivo AppModule, por
lo que Dagger se encarga de crear este método*/
```

```
class MainViewModel @ViewModelInject constructor(
    val mainRepository: MainRepository
): ViewModel(){
    /*Observador que monitorea cambios en la base de datos a través del repositorio, que es una capa
    intermedia entre la base de datos y la aplicación*/
    //ACOMODA LOS RECORRIDOS POR FECHA
    private val runSortedByDate = mainRepository.getAllRunsSortedByDate()
    //ACOMODA LOS RECORRIDOS POR FECHA
    private val runSortedByDistance = mainRepository.getAllRunsSortedByDistance()
    //ACOMODA LOS RECORRIDOS POR FECHA
    private val runSortedByCaloriesBurned = mainRepository.getAllRunsSortedByCaloriesBurned()
    //ACOMODA LOS RECORRIDOS POR FECHA
    private val runSortedByTimeInMillis = mainRepository.getAllRunsSortedByTimeInMillis()
    //ACOMODA LOS RECORRIDOS POR FECHA
    private val runSortedByAvgSpeed = mainRepository.getAllRunsSortedByAvgSpeed()

    /*Se creara un tipo especial de dato llamado MediatorLiveData usado para obtener distintos datos
    recolectados y juntarlos en un tipo de paquetito para poderlos manejar todos a la vez, esto se
    va a hacer para manejar todos los datos que tenemos arriba en una sola variable llamada runs
    que maneje un tipo de dato lista que contenga datos de tipo Run, que son los que están asociados
    a la base de datos*/
    val runs = MediatorLiveData<List<Run>>()
    /*Se crean varias variables que mandan a llamar las constantes de la clase ENUM llamada SortType
    dentro de la variable solo se mandará a llamar la principal que es DATE*/
    var sortType = SortType.DATE

    /*En el bloque init se puede unir todos los datos obtenidos de la base de datos para solo emitir
    los valores correspondientes que coincidan con el sortType correspondiente*/
    init {
        /*Cuando que sea emitido un dato de tipo Date hacia el objeto run que está instanciando la
        clase MediatorLiveData por lo que tiene unidos todos los valores de la la base de datos, el
        observador es llamado y checa si el sortType coincide con su correspondiente constante
        proveniente de la clase ENUM llamada SortType para mostrar un dato u otro*/
        //MOSTRAR EL DATO DE FECHA PROVENIENTE DEL OBJETO RUN QUE JUNTA TODOS LOS TIPOS DE DATO
        runs.addSource(runSortedByDate){ result ->
            if(sortType == SortType.DATE){
                result.let { runs.value = it }
            }
        }
        //MOSTRAR EL DATO DE VELPROM PROVENIENTE DEL OBJETO RUN QUE JUNTA TODOS LOS TIPOS DE DATO
        runs.addSource(runSortedByAvgSpeed){ result ->
            if(sortType == SortType.AVG_SPEED){
                result.let { runs.value = it }
            }
        }
        //MOSTRAR EL DATO DE CALORÍAS QUEMADAS PROVENIENTE DEL OBJETO RUN QUE JUNTA TODOS LOS TIPOS
DE DATO
        runs.addSource(runSortedByCaloriesBurned){ result ->
            if(sortType == SortType.CALORIES_BURNED){
                result.let { runs.value = it }
            }
        }
        //MOSTRAR EL DATO DE DITANCIA PROVENIENTE DEL OBJETO RUN QUE JUNTA TODOS LOS TIPOS DE DATO
        runs.addSource(runSortedByDistance){ result ->
            if(sortType == SortType.DISTANCE){
                result.let { runs.value = it }
            }
        }
        //MOSTRAR EL DATO DE TIEMPO EN MILISEGUNDOS PROVENIENTE DEL OBJETO RUN QUE JUNTA TODOS LOS
TIPOS DE DATO
        runs.addSource(runSortedByTimeInMillis){ result ->
            if(sortType == SortType.RUNNING_TIME){
                result.let { runs.value = it }
```

```
            }
        }
    }

    /*Función para asignar cada sortType a cada tipo de dato*/
    fun sortRuns(sortType: SortType) = when(sortType){
        SortType.DATE -> runSortedByDate.value?.let { runs.value = it }
        SortType.RUNNING_TIME -> runSortedByTimeInMillis.value?.let { runs.value = it }
        SortType.AVG_SPEED -> runSortedByAvgSpeed.value?.let { runs.value = it }
        SortType.DISTANCE -> runSortedByDistance.value?.let { runs.value = it }
        SortType.CALORIES_BURNED -> runSortedByCaloriesBurned.value?.let { runs.value = it }
    }.also {
        this.sortType = sortType
    }

    /*Función para insertar los datos recopilados del recorrido en la clase Run en forma de
corrutina
    para que se ejecute en paralelo con otras funciones de la aplicación*/
    fun insertRun(run: Run) = viewModelScope.launch {
        mainRepository.insertRun(run)
    }
}
```

StatisticsViewModel

```
package com.androiddevs.runningappyt.ui.viewmodels
import androidx.hilt.lifecycle.ViewModelInject
import androidx.lifecycle.ViewModel
import com.androiddevs.runningappyt.repositories.MainRepository

/*El propósito del StatisticsViewModel es recolectar los datos recolectados del repositorio y
proveerlos para los fragmentos que los mostrarán en pantalla, los fragmentos son pedazos de la
interfaz en la aplicación Android que sirven para hacer que distintos datos dinámicos se puedan
imprimir en una misma zona de la aplicación. La clase MainViewModel y StatisticsViewModel usan el
mismo repositorio*/

/*Para poder recolectar los datos del repositorio es necesario usar la anotación @ViewModelInject()
para crear una fábrica de ViewModels, esto se hace creando un objeto MainRepository que es instancia
de la clase MainRepository que es donde se creó el repositorio del proyecto que recolecta los datos
para pasárselos a la base de datos y a la interfaz de usuario Android, aunque no se puso
explícitamente un método para dar datos del repositorio, esto se puede hacer por medio de Dagger ya
que el parámetro de la función creada en MainRepository recibe como parámetro un objeto RunDAO de la
interfaz con el mismo nombre, esto fue descrito para la librería Dagger en el archivo AppModule, por
lo que Dagger se encarga de crear este método*/
class StatisticsViewModel @ViewModelInject constructor(
    val mainRepository: MainRepository
): ViewModel(){
    /*Dentro de esta clase se crearán las gráficas estadísticas de los recorridos*/
    val totalTimeRun = mainRepository.getTotalTimeInMillis()
    val totalDistance = mainRepository.getTotalDistance()
    val totalCaloriesBurned = mainRepository.getTotalCaloriesBurned()
    val totalAvgSpeed = mainRepository.getTotalAvgSpeed()

    val runsSortedByDate = mainRepository.getAllRunsSortedByDate()
}
```

BaseApplication

```
package com.androiddevs.runningappyt
import android.app.Application
```

```
import dagger.hilt.android.HiltAndroidApp
import timber.log.Timber

/*La clase BaseApplication hereda de la clase Application y usa la anotación @HiltAndroidApp para
que se pueda utilizar la librería Dagger Hilt y que haga inyección de dependencias a las distintas
clases que lo necesitan en segundo plano*/
@HiltAndroidApp
class BaseApplication : Application() {
    /*Cuando la aplicación sea iniciada, se ejecutará esta acción usando la librería Timber, la
    librería Timber sirve para que el usuario pueda registrarse en la aplicación, osea hacer su
    Login de una forma más sencilla*/
    override fun onCreate() {
        super.onCreate()
        Timber.plant(Timber.DebugTree())
    }
}

/*Al dar clic en el botón de Build -> Rebuild Project, se crearán una serie de archivos Java que
* permiten agregar las Dependency Injections*/
```

ExampleInstrumentedTest

```
package com.androiddevs.runningappyt

import androidx.test.platform.app.InstrumentationRegistry
import androidx.test.ext.junit.runners.AndroidJUnit4

import org.junit.Test
import org.junit.runner.RunWith

import org.junit.Assert.*

/**
 * Instrumented test, which will execute on an Android device.
 *
 * See [testing documentation](http://d.android.com/tools/testing).
 */
@RunWith(AndroidJUnit4::class)
class ExampleInstrumentedTest {
    @Test
    fun useAppContext() {
        // Context of the app under test.
        val appContext = InstrumentationRegistry.getInstrumentation().targetContext
        assertEquals("com.androiddevs.runningappyt", appContext.packageName)
    }
}
```

ExampleUnitTest

```
package com.androiddevs.runningappyt

import org.junit.Test

import org.junit.Assert.*

/**
 * Example local unit test, which will execute on the development machine (host).
 *
 * See [testing documentation](http://d.android.com/tools/testing).
 */
class ExampleUnitTest {
    @Test
    fun addition_isCorrect() {
        assertEquals(4, 2 + 2)
```

```
    }
}
```

BuildConfig

```
/**
 * Automatically generated file. DO NOT MODIFY
 */
package com.androiddevs.runningappyt;

public final class BuildConfig {
  public static final boolean DEBUG = Boolean.parseBoolean("true");
  public static final String APPLICATION_ID = "com.androiddevs.runningappyt";
  public static final String BUILD_TYPE = "debug";
  public static final int VERSION_CODE = 1;
  public static final String VERSION_NAME = "1.0";
}
```

RunDAO_ImpI

```
package com.androiddevs.runningappyt.db;

import android.database.Cursor;
import android.graphics.Bitmap;
import androidx.lifecycle.LiveData;
import androidx.room.CoroutinesRoom;
import androidx.room.EntityDeletionOrUpdateAdapter;
import androidx.room.EntityInsertionAdapter;
import androidx.room.RoomDatabase;
import androidx.room.RoomSQLiteQuery;
import androidx.room.util.CursorUtil;
import androidx.room.util.DBUtil;
import androidx.sqlite.db.SupportSQLiteStatement;
import java.lang.Exception;
import java.lang.Float;
import java.lang.Integer;
import java.lang.Long;
import java.lang.Object;
import java.lang.Override;
import java.lang.String;
import java.lang.SuppressWarnings;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.Callable;
import javax.annotation.Generated;
import kotlin.Unit;
import kotlin.coroutines.Continuation;

@Generated("androidx.room.RoomProcessor")
@SuppressWarnings({"unchecked", "deprecation"})
public final class RunDAO_Impl implements RunDAO {
  private final RoomDatabase __db;

  private final EntityInsertionAdapter<Run> __insertionAdapterOfRun;

  private final Converters __converters = new Converters();

  private final EntityDeletionOrUpdateAdapter<Run> __deletionAdapterOfRun;

  public RunDAO_Impl(RoomDatabase __db) {
    this.__db = __db;
    this.__insertionAdapterOfRun = new EntityInsertionAdapter<Run>(__db) {
      @Override
      public String createQuery() {
```

```
        return "INSERT OR REPLACE INTO `running_table`
(`id`,`img`,`timestamp`,`avgSpeedInKNMH`,`distanceInMeters`,`timeInMillis`,`caloriesBurned`) VALUES
(?,?,?,?,?,?,?)";
      }

      @Override
      public void bind(SupportSQLiteStatement stmt, Run value) {
        if (value.getId() == null) {
          stmt.bindNull(1);
        } else {
          stmt.bindLong(1, value.getId());
        }
        final byte[] _tmp;
        _tmp = __converters.fromBitmap(value.getImg());
        if (_tmp == null) {
          stmt.bindNull(2);
        } else {
          stmt.bindBlob(2, _tmp);
        }
        stmt.bindLong(3, value.getTimestamp());
        stmt.bindDouble(4, value.getAvgSpeedInKNMH());
        stmt.bindLong(5, value.getDistanceInMeters());
        stmt.bindLong(6, value.getTimeInMillis());
        stmt.bindLong(7, value.getCaloriesBurned());
      }
    };
    this.__deletionAdapterOfRun = new EntityDeletionOrUpdateAdapter<Run>(__db) {
      @Override
      public String createQuery() {
        return "DELETE FROM `running_table` WHERE `id` = ?";
      }

      @Override
      public void bind(SupportSQLiteStatement stmt, Run value) {
        if (value.getId() == null) {
          stmt.bindNull(1);
        } else {
          stmt.bindLong(1, value.getId());
        }
      }
    };
  }

  @Override
  public Object insertRun(final Run run, final Continuation<? super Unit> p1) {
    return CoroutinesRoom.execute(__db, true, new Callable<Unit>() {
      @Override
      public Unit call() throws Exception {
        __db.beginTransaction();
        try {
          __insertionAdapterOfRun.insert(run);
          __db.setTransactionSuccessful();
          return Unit.INSTANCE;
        } finally {
          __db.endTransaction();
        }
      }
    }, p1);
  }

  @Override
  public Object deleteRun(final Run run, final Continuation<? super Unit> p1) {
    return CoroutinesRoom.execute(__db, true, new Callable<Unit>() {
```

```
      @Override
      public Unit call() throws Exception {
        __db.beginTransaction();
        try {
          __deletionAdapterOfRun.handle(run);
          __db.setTransactionSuccessful();
          return Unit.INSTANCE;
        } finally {
          __db.endTransaction();
        }
      }
    }, p1);
  }

  @Override
  public LiveData<List<Run>> getAllRunsSortedByDate() {
    final String _sql = "SELECT * FROM running_table ORDER BY timestamp DESC";
    final RoomSQLiteQuery _statement = RoomSQLiteQuery.acquire(_sql, 0);
    return __db.getInvalidationTracker().createLiveData(new String[]{"running_table"}, false, new
Callable<List<Run>>() {
      @Override
      public List<Run> call() throws Exception {
        final Cursor _cursor = DBUtil.query(__db, _statement, false, null);
        try {
          final int _cursorIndexOfId = CursorUtil.getColumnIndexOrThrow(_cursor, "id");
          final int _cursorIndexOfImg = CursorUtil.getColumnIndexOrThrow(_cursor, "img");
          final int _cursorIndexOfTimestamp = CursorUtil.getColumnIndexOrThrow(_cursor,
"timestamp");
          final int _cursorIndexOfAvgSpeedInKNMH = CursorUtil.getColumnIndexOrThrow(_cursor,
"avgSpeedInKNMH");
          final int _cursorIndexOfDistanceInMeters = CursorUtil.getColumnIndexOrThrow(_cursor,
"distanceInMeters");
          final int _cursorIndexOfTimeInMillis = CursorUtil.getColumnIndexOrThrow(_cursor,
"timeInMillis");
          final int _cursorIndexOfCaloriesBurned = CursorUtil.getColumnIndexOrThrow(_cursor,
"caloriesBurned");
          final List<Run> _result = new ArrayList<Run>(_cursor.getCount());
          while(_cursor.moveToNext()) {
            final Run _item;
            final Bitmap _tmpImg;
            final byte[] _tmp;
            _tmp = _cursor.getBlob(_cursorIndexOfImg);
            _tmpImg = __converters.toBitMap(_tmp);
            final long _tmpTimestamp;
            _tmpTimestamp = _cursor.getLong(_cursorIndexOfTimestamp);
            final float _tmpAvgSpeedInKNMH;
            _tmpAvgSpeedInKNMH = _cursor.getFloat(_cursorIndexOfAvgSpeedInKNMH);
            final int _tmpDistanceInMeters;
            _tmpDistanceInMeters = _cursor.getInt(_cursorIndexOfDistanceInMeters);
            final long _tmpTimeInMillis;
            _tmpTimeInMillis = _cursor.getLong(_cursorIndexOfTimeInMillis);
            final int _tmpCaloriesBurned;
            _tmpCaloriesBurned = _cursor.getInt(_cursorIndexOfCaloriesBurned);
            _item = new
Run(_tmpImg,_tmpTimestamp,_tmpAvgSpeedInKNMH,_tmpDistanceInMeters,_tmpTimeInMillis,_tmpCaloriesBurne
d);
            final Integer _tmpId;
            if (_cursor.isNull(_cursorIndexOfId)) {
              _tmpId = null;
            } else {
              _tmpId = _cursor.getInt(_cursorIndexOfId);
            }
            _item.setId(_tmpId);
```

```
            _result.add(_item);
          }
          return _result;
        } finally {
          _cursor.close();
        }
      }

      @Override
      protected void finalize() {
        _statement.release();
      }
    });
  }

  @Override
  public LiveData<List<Run>> getAllRunsSortedByTimeInMillis() {
    final String _sql = "SELECT * FROM running_table ORDER BY timeInMillis DESC";
    final RoomSQLiteQuery _statement = RoomSQLiteQuery.acquire(_sql, 0);
    return __db.getInvalidationTracker().createLiveData(new String[]{"running_table"}, false, new
Callable<List<Run>>() {
      @Override
      public List<Run> call() throws Exception {
        final Cursor _cursor = DBUtil.query(__db, _statement, false, null);
        try {
          final int _cursorIndexOfId = CursorUtil.getColumnIndexOrThrow(_cursor, "id");
          final int _cursorIndexOfImg = CursorUtil.getColumnIndexOrThrow(_cursor, "img");
          final int _cursorIndexOfTimestamp = CursorUtil.getColumnIndexOrThrow(_cursor,
"timestamp");
          final int _cursorIndexOfAvgSpeedInKNMH = CursorUtil.getColumnIndexOrThrow(_cursor,
"avgSpeedInKNMH");
          final int _cursorIndexOfDistanceInMeters = CursorUtil.getColumnIndexOrThrow(_cursor,
"distanceInMeters");
          final int _cursorIndexOfTimeInMillis = CursorUtil.getColumnIndexOrThrow(_cursor,
"timeInMillis");
          final int _cursorIndexOfCaloriesBurned = CursorUtil.getColumnIndexOrThrow(_cursor,
"caloriesBurned");
          final List<Run> _result = new ArrayList<Run>(_cursor.getCount());
          while(_cursor.moveToNext()) {
            final Run _item;
            final Bitmap _tmpImg;
            final byte[] _tmp;
            _tmp = _cursor.getBlob(_cursorIndexOfImg);
            _tmpImg = __converters.toBitMap(_tmp);
            final long _tmpTimestamp;
            _tmpTimestamp = _cursor.getLong(_cursorIndexOfTimestamp);
            final float _tmpAvgSpeedInKNMH;
            _tmpAvgSpeedInKNMH = _cursor.getFloat(_cursorIndexOfAvgSpeedInKNMH);
            final int _tmpDistanceInMeters;
            _tmpDistanceInMeters = _cursor.getInt(_cursorIndexOfDistanceInMeters);
            final long _tmpTimeInMillis;
            _tmpTimeInMillis = _cursor.getLong(_cursorIndexOfTimeInMillis);
            final int _tmpCaloriesBurned;
            _tmpCaloriesBurned = _cursor.getInt(_cursorIndexOfCaloriesBurned);
            _item = new
Run(_tmpImg,_tmpTimestamp,_tmpAvgSpeedInKNMH,_tmpDistanceInMeters,_tmpTimeInMillis,_tmpCaloriesBurne
d);
            final Integer _tmpId;
            if (_cursor.isNull(_cursorIndexOfId)) {
              _tmpId = null;
            } else {
              _tmpId = _cursor.getInt(_cursorIndexOfId);
            }
```

```
            _item.setId(_tmpId);
            _result.add(_item);
          }
          return _result;
        } finally {
          _cursor.close();
        }
      }

      @Override
      protected void finalize() {
        _statement.release();
      }
    });
  }

  @Override
  public LiveData<List<Run>> getAllRunsSortedByCaloriesBurned() {
    final String _sql = "SELECT * FROM running_table ORDER BY caloriesBurned DESC";
    final RoomSQLiteQuery _statement = RoomSQLiteQuery.acquire(_sql, 0);
    return __db.getInvalidationTracker().createLiveData(new String[]{"running_table"}, false, new
Callable<List<Run>>() {
      @Override
      public List<Run> call() throws Exception {
        final Cursor _cursor = DBUtil.query(__db, _statement, false, null);
        try {
          final int _cursorIndexOfId = CursorUtil.getColumnIndexOrThrow(_cursor, "id");
          final int _cursorIndexOfImg = CursorUtil.getColumnIndexOrThrow(_cursor, "img");
          final int _cursorIndexOfTimestamp = CursorUtil.getColumnIndexOrThrow(_cursor,
"timestamp");
          final int _cursorIndexOfAvgSpeedInKNMH = CursorUtil.getColumnIndexOrThrow(_cursor,
"avgSpeedInKNMH");
          final int _cursorIndexOfDistanceInMeters = CursorUtil.getColumnIndexOrThrow(_cursor,
"distanceInMeters");
          final int _cursorIndexOfTimeInMillis = CursorUtil.getColumnIndexOrThrow(_cursor,
"timeInMillis");
          final int _cursorIndexOfCaloriesBurned = CursorUtil.getColumnIndexOrThrow(_cursor,
"caloriesBurned");
          final List<Run> _result = new ArrayList<Run>(_cursor.getCount());
          while(_cursor.moveToNext()) {
            final Run _item;
            final Bitmap _tmpImg;
            final byte[] _tmp;
            _tmp = _cursor.getBlob(_cursorIndexOfImg);
            _tmpImg = __converters.toBitMap(_tmp);
            final long _tmpTimestamp;
            _tmpTimestamp = _cursor.getLong(_cursorIndexOfTimestamp);
            final float _tmpAvgSpeedInKNMH;
            _tmpAvgSpeedInKNMH = _cursor.getFloat(_cursorIndexOfAvgSpeedInKNMH);
            final int _tmpDistanceInMeters;
            _tmpDistanceInMeters = _cursor.getInt(_cursorIndexOfDistanceInMeters);
            final long _tmpTimeInMillis;
            _tmpTimeInMillis = _cursor.getLong(_cursorIndexOfTimeInMillis);
            final int _tmpCaloriesBurned;
            _tmpCaloriesBurned = _cursor.getInt(_cursorIndexOfCaloriesBurned);
            _item = new
Run(_tmpImg,_tmpTimestamp,_tmpAvgSpeedInKNMH,_tmpDistanceInMeters,_tmpTimeInMillis,_tmpCaloriesBurne
d);
            final Integer _tmpId;
            if (_cursor.isNull(_cursorIndexOfId)) {
              _tmpId = null;
            } else {
              _tmpId = _cursor.getInt(_cursorIndexOfId);
```

```
            }
            _item.setId(_tmpId);
            _result.add(_item);
          }
          return _result;
        } finally {
          _cursor.close();
        }
      }

      @Override
      protected void finalize() {
        _statement.release();
      }
    });
  }

  @Override
  public LiveData<List<Run>> getAllRunsSortedByAvgSpeed() {
    final String _sql = "SELECT * FROM running_table ORDER BY avgSpeedInKNMH DESC";
    final RoomSQLiteQuery _statement = RoomSQLiteQuery.acquire(_sql, 0);
    return __db.getInvalidationTracker().createLiveData(new String[]{"running_table"}, false, new
Callable<List<Run>>() {
      @Override
      public List<Run> call() throws Exception {
        final Cursor _cursor = DBUtil.query(__db, _statement, false, null);
        try {
          final int _cursorIndexOfId = CursorUtil.getColumnIndexOrThrow(_cursor, "id");
          final int _cursorIndexOfImg = CursorUtil.getColumnIndexOrThrow(_cursor, "img");
          final int _cursorIndexOfTimestamp = CursorUtil.getColumnIndexOrThrow(_cursor,
"timestamp");
          final int _cursorIndexOfAvgSpeedInKNMH = CursorUtil.getColumnIndexOrThrow(_cursor,
"avgSpeedInKNMH");
          final int _cursorIndexOfDistanceInMeters = CursorUtil.getColumnIndexOrThrow(_cursor,
"distanceInMeters");
          final int _cursorIndexOfTimeInMillis = CursorUtil.getColumnIndexOrThrow(_cursor,
"timeInMillis");
          final int _cursorIndexOfCaloriesBurned = CursorUtil.getColumnIndexOrThrow(_cursor,
"caloriesBurned");
          final List<Run> _result = new ArrayList<Run>(_cursor.getCount());
          while(_cursor.moveToNext()) {
            final Run _item;
            final Bitmap _tmpImg;
            final byte[] _tmp;
            _tmp = _cursor.getBlob(_cursorIndexOfImg);
            _tmpImg = __converters.toBitMap(_tmp);
            final long _tmpTimestamp;
            _tmpTimestamp = _cursor.getLong(_cursorIndexOfTimestamp);
            final float _tmpAvgSpeedInKNMH;
            _tmpAvgSpeedInKNMH = _cursor.getFloat(_cursorIndexOfAvgSpeedInKNMH);
            final int _tmpDistanceInMeters;
            _tmpDistanceInMeters = _cursor.getInt(_cursorIndexOfDistanceInMeters);
            final long _tmpTimeInMillis;
            _tmpTimeInMillis = _cursor.getLong(_cursorIndexOfTimeInMillis);
            final int _tmpCaloriesBurned;
            _tmpCaloriesBurned = _cursor.getInt(_cursorIndexOfCaloriesBurned);
            _item = new
Run(_tmpImg,_tmpTimestamp,_tmpAvgSpeedInKNMH,_tmpDistanceInMeters,_tmpTimeInMillis,_tmpCaloriesBurne
d);
            final Integer _tmpId;
            if (_cursor.isNull(_cursorIndexOfId)) {
              _tmpId = null;
            } else {
```

```
            _tmpId = _cursor.getInt(_cursorIndexOfId);
          }
          _item.setId(_tmpId);
          _result.add(_item);
        }
        return _result;
      } finally {
        _cursor.close();
      }
    }

    @Override
    protected void finalize() {
      _statement.release();
    }
  });
}

@Override
public LiveData<List<Run>> getAllRunsSortedByDistance() {
  final String _sql = "SELECT * FROM running_table ORDER BY distanceInMeters DESC";
  final RoomSQLiteQuery _statement = RoomSQLiteQuery.acquire(_sql, 0);
  return __db.getInvalidationTracker().createLiveData(new String[]{"running_table"}, false, new
Callable<List<Run>>() {
    @Override
    public List<Run> call() throws Exception {
      final Cursor _cursor = DBUtil.query(__db, _statement, false, null);
      try {
        final int _cursorIndexOfId = CursorUtil.getColumnIndexOrThrow(_cursor, "id");
        final int _cursorIndexOfImg = CursorUtil.getColumnIndexOrThrow(_cursor, "img");
        final int _cursorIndexOfTimestamp = CursorUtil.getColumnIndexOrThrow(_cursor,
"timestamp");
        final int _cursorIndexOfAvgSpeedInKNMH = CursorUtil.getColumnIndexOrThrow(_cursor,
"avgSpeedInKNMH");
        final int _cursorIndexOfDistanceInMeters = CursorUtil.getColumnIndexOrThrow(_cursor,
"distanceInMeters");
        final int _cursorIndexOfTimeInMillis = CursorUtil.getColumnIndexOrThrow(_cursor,
"timeInMillis");
        final int _cursorIndexOfCaloriesBurned = CursorUtil.getColumnIndexOrThrow(_cursor,
"caloriesBurned");
        final List<Run> _result = new ArrayList<Run>(_cursor.getCount());
        while(_cursor.moveToNext()) {
          final Run _item;
          final Bitmap _tmpImg;
          final byte[] _tmp;
          _tmp = _cursor.getBlob(_cursorIndexOfImg);
          _tmpImg = __converters.toBitMap(_tmp);
          final long _tmpTimestamp;
          _tmpTimestamp = _cursor.getLong(_cursorIndexOfTimestamp);
          final float _tmpAvgSpeedInKNMH;
          _tmpAvgSpeedInKNMH = _cursor.getFloat(_cursorIndexOfAvgSpeedInKNMH);
          final int _tmpDistanceInMeters;
          _tmpDistanceInMeters = _cursor.getInt(_cursorIndexOfDistanceInMeters);
          final long _tmpTimeInMillis;
          _tmpTimeInMillis = _cursor.getLong(_cursorIndexOfTimeInMillis);
          final int _tmpCaloriesBurned;
          _tmpCaloriesBurned = _cursor.getInt(_cursorIndexOfCaloriesBurned);
          _item = new
Run(_tmpImg,_tmpTimestamp,_tmpAvgSpeedInKNMH,_tmpDistanceInMeters,_tmpTimeInMillis,_tmpCaloriesBurne
d);
          final Integer _tmpId;
          if (_cursor.isNull(_cursorIndexOfId)) {
            _tmpId = null;
```

```
            } else {
              _tmpId = _cursor.getInt(_cursorIndexOfId);
            }
            _item.setId(_tmpId);
            _result.add(_item);
          }
          return _result;
        } finally {
          _cursor.close();
        }
      }

      @Override
      protected void finalize() {
        _statement.release();
      }
    });
  }

  @Override
  public LiveData<Long> getTotalTimeInMillis() {
    final String _sql = "SELECT SUM(timeInMillis) FROM running_table";
    final RoomSQLiteQuery _statement = RoomSQLiteQuery.acquire(_sql, 0);
    return __db.getInvalidationTracker().createLiveData(new String[]{"running_table"}, false, new
Callable<Long>() {
      @Override
      public Long call() throws Exception {
        final Cursor _cursor = DBUtil.query(__db, _statement, false, null);
        try {
          final Long _result;
          if(_cursor.moveToFirst()) {
            final Long _tmp;
            if (_cursor.isNull(0)) {
              _tmp = null;
            } else {
              _tmp = _cursor.getLong(0);
            }
            _result = _tmp;
          } else {
            _result = null;
          }
          return _result;
        } finally {
          _cursor.close();
        }
      }

      @Override
      protected void finalize() {
        _statement.release();
      }
    });
  }

  @Override
  public LiveData<Integer> getTotalCaloriesBurned() {
    final String _sql = "SELECT SUM(caloriesBurned) FROM running_table";
    final RoomSQLiteQuery _statement = RoomSQLiteQuery.acquire(_sql, 0);
    return __db.getInvalidationTracker().createLiveData(new String[]{"running_table"}, false, new
Callable<Integer>() {
      @Override
      public Integer call() throws Exception {
        final Cursor _cursor = DBUtil.query(__db, _statement, false, null);
```

```
        try {
          final Integer _result;
          if(_cursor.moveToFirst()) {
            final Integer _tmp;
            if (_cursor.isNull(0)) {
              _tmp = null;
            } else {
              _tmp = _cursor.getInt(0);
            }
            _result = _tmp;
          } else {
            _result = null;
          }
          return _result;
        } finally {
          _cursor.close();
        }
      }

      @Override
      protected void finalize() {
        _statement.release();
      }
    });
  }

  @Override
  public LiveData<Integer> getTotalDistance() {
    final String _sql = "SELECT SUM(distanceInMeters) FROM running_table";
    final RoomSQLiteQuery _statement = RoomSQLiteQuery.acquire(_sql, 0);
    return __db.getInvalidationTracker().createLiveData(new String[]{"running_table"}, false, new
Callable<Integer>() {
      @Override
      public Integer call() throws Exception {
        final Cursor _cursor = DBUtil.query(__db, _statement, false, null);
        try {
          final Integer _result;
          if(_cursor.moveToFirst()) {
            final Integer _tmp;
            if (_cursor.isNull(0)) {
              _tmp = null;
            } else {
              _tmp = _cursor.getInt(0);
            }
            _result = _tmp;
          } else {
            _result = null;
          }
          return _result;
        } finally {
          _cursor.close();
        }
      }

      @Override
      protected void finalize() {
        _statement.release();
      }
    });
  }

  @Override
  public LiveData<Float> getTotalAvgSpeed() {
```

```
    final String _sql = "SELECT AVG(avgSpeedInKNMH) FROM running_table";
    final RoomSQLiteQuery _statement = RoomSQLiteQuery.acquire(_sql, 0);
    return __db.getInvalidationTracker().createLiveData(new String[]{"running_table"}, false, new
Callable<Float>() {
      @Override
      public Float call() throws Exception {
        final Cursor _cursor = DBUtil.query(__db, _statement, false, null);
        try {
          final Float _result;
          if(_cursor.moveToFirst()) {
            final Float _tmp;
            if (_cursor.isNull(0)) {
              _tmp = null;
            } else {
              _tmp = _cursor.getFloat(0);
            }
            _result = _tmp;
          } else {
            _result = null;
          }
          return _result;
        } finally {
          _cursor.close();
        }
      }

      @Override
      protected void finalize() {
        _statement.release();
      }
    });
  }
}
```

RunnimgDatabase_ImpI

```
package com.androiddevs.runningappyt.db;

import androidx.room.DatabaseConfiguration;
import androidx.room.InvalidationTracker;
import androidx.room.RoomOpenHelper;
import androidx.room.RoomOpenHelper.Delegate;
import androidx.room.RoomOpenHelper.ValidationResult;
import androidx.room.util.DBUtil;
import androidx.room.util.TableInfo;
import androidx.room.util.TableInfo.Column;
import androidx.room.util.TableInfo.ForeignKey;
import androidx.room.util.TableInfo.Index;
import androidx.sqlite.db.SupportSQLiteDatabase;
import androidx.sqlite.db.SupportSQLiteOpenHelper;
import androidx.sqlite.db.SupportSQLiteOpenHelper.Callback;
import androidx.sqlite.db.SupportSQLiteOpenHelper.Configuration;
import java.lang.Override;
import java.lang.String;
import java.lang.SuppressWarnings;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Set;
import javax.annotation.Generated;

@Generated("androidx.room.RoomProcessor")
@SuppressWarnings({"unchecked", "deprecation"})
public final class RunningDatabase_Impl extends RunningDatabase {
  private volatile RunDAO _runDAO;
```

```
  @Override
  protected SupportSQLiteOpenHelper createOpenHelper(DatabaseConfiguration configuration) {
    final SupportSQLiteOpenHelper.Callback _openCallback = new RoomOpenHelper(configuration, new
RoomOpenHelper.Delegate(1) {
      @Override
      public void createAllTables(SupportSQLiteDatabase _db) {
        _db.execSQL("CREATE TABLE IF NOT EXISTS `running_table` (`id` INTEGER PRIMARY KEY
AUTOINCREMENT, `img` BLOB, `timestamp` INTEGER NOT NULL, `avgSpeedInKNMH` REAL NOT NULL,
`distanceInMeters` INTEGER NOT NULL, `timeInMillis` INTEGER NOT NULL, `caloriesBurned` INTEGER NOT
NULL)");
        _db.execSQL("CREATE TABLE IF NOT EXISTS room_master_table (id INTEGER PRIMARY
KEY,identity_hash TEXT)");
        _db.execSQL("INSERT OR REPLACE INTO room_master_table (id,identity_hash) VALUES(42,
'1c90b2658769956d6a2c61fe8adb40e2')");
      }

      @Override
      public void dropAllTables(SupportSQLiteDatabase _db) {
        _db.execSQL("DROP TABLE IF EXISTS `running_table`");
        if (mCallbacks != null) {
          for (int _i = 0, _size = mCallbacks.size(); _i < _size; _i++) {
            mCallbacks.get(_i).onDestructiveMigration(_db);
          }
        }
      }

      @Override
      protected void onCreate(SupportSQLiteDatabase _db) {
        if (mCallbacks != null) {
          for (int _i = 0, _size = mCallbacks.size(); _i < _size; _i++) {
            mCallbacks.get(_i).onCreate(_db);
          }
        }
      }

      @Override
      public void onOpen(SupportSQLiteDatabase _db) {
        mDatabase = _db;
        internalInitInvalidationTracker(_db);
        if (mCallbacks != null) {
          for (int _i = 0, _size = mCallbacks.size(); _i < _size; _i++) {
            mCallbacks.get(_i).onOpen(_db);
          }
        }
      }

      @Override
      public void onPreMigrate(SupportSQLiteDatabase _db) {
        DBUtil.dropFtsSyncTriggers(_db);
      }

      @Override
      public void onPostMigrate(SupportSQLiteDatabase _db) {
      }

      @Override
      protected RoomOpenHelper.ValidationResult onValidateSchema(SupportSQLiteDatabase _db) {
        final HashMap<String, TableInfo.Column> _columnsRunningTable = new HashMap<String,
TableInfo.Column>(7);
        _columnsRunningTable.put("id", new TableInfo.Column("id", "INTEGER", false, 1, null,
TableInfo.CREATED_FROM_ENTITY));
        _columnsRunningTable.put("img", new TableInfo.Column("img", "BLOB", false, 0, null,
```

```
TableInfo.CREATED_FROM_ENTITY));
        _columnsRunningTable.put("timestamp", new TableInfo.Column("timestamp", "INTEGER", true, 0,
null, TableInfo.CREATED_FROM_ENTITY));
        _columnsRunningTable.put("avgSpeedInKNMH", new TableInfo.Column("avgSpeedInKNMH", "REAL",
true, 0, null, TableInfo.CREATED_FROM_ENTITY));
        _columnsRunningTable.put("distanceInMeters", new TableInfo.Column("distanceInMeters",
"INTEGER", true, 0, null, TableInfo.CREATED_FROM_ENTITY));
        _columnsRunningTable.put("timeInMillis", new TableInfo.Column("timeInMillis", "INTEGER",
true, 0, null, TableInfo.CREATED_FROM_ENTITY));
        _columnsRunningTable.put("caloriesBurned", new TableInfo.Column("caloriesBurned", "INTEGER",
true, 0, null, TableInfo.CREATED_FROM_ENTITY));
        final HashSet<TableInfo.ForeignKey> _foreignKeysRunningTable = new
HashSet<TableInfo.ForeignKey>(0);
        final HashSet<TableInfo.Index> _indicesRunningTable = new HashSet<TableInfo.Index>(0);
        final TableInfo _infoRunningTable = new TableInfo("running_table", _columnsRunningTable,
_foreignKeysRunningTable, _indicesRunningTable);
        final TableInfo _existingRunningTable = TableInfo.read(_db, "running_table");
        if (! _infoRunningTable.equals(_existingRunningTable)) {
          return new RoomOpenHelper.ValidationResult(false,
"running_table(com.androiddevs.runningappyt.db.Run).\n"
                  + " Expected:\n" + _infoRunningTable + "\n"
                  + " Found:\n" + _existingRunningTable);
        }
        return new RoomOpenHelper.ValidationResult(true, null);
      }
    }, "1c90b2658769956d6a2c61fe8adb40e2", "016b64eb23ee52a53dcebcc429117b6a");
    final SupportSQLiteOpenHelper.Configuration _sqliteConfig =
SupportSQLiteOpenHelper.Configuration.builder(configuration.context)
        .name(configuration.name)
        .callback(_openCallback)
        .build();
    final SupportSQLiteOpenHelper _helper =
configuration.sqliteOpenHelperFactory.create(_sqliteConfig);
    return _helper;
  }

  @Override
  protected InvalidationTracker createInvalidationTracker() {
    final HashMap<String, String> _shadowTablesMap = new HashMap<String, String>(0);
    HashMap<String, Set<String>> _viewTables = new HashMap<String, Set<String>>(0);
    return new InvalidationTracker(this, _shadowTablesMap, _viewTables, "running_table");
  }

  @Override
  public void clearAllTables() {
    super.assertNotMainThread();
    final SupportSQLiteDatabase _db = super.getOpenHelper().getWritableDatabase();
    try {
      super.beginTransaction();
      _db.execSQL("DELETE FROM `running_table`");
      super.setTransactionSuccessful();
    } finally {
      super.endTransaction();
      _db.query("PRAGMA wal_checkpoint(FULL)").close();
      if (!_db.inTransaction()) {
        _db.execSQL("VACUUM");
      }
    }
  }

  @Override
  public RunDAO getRunDAO() {
    if (_runDAO != null) {
```

```
      return _runDAO;
    } else {
      synchronized(this) {
        if(_runDAO == null) {
          _runDAO = new RunDAO_Impl(this);
        }
        return _runDAO;
      }
    }
  }
}
```

AppModule_ProvideFirsTimeToggleFactory

```
package com.androiddevs.runningappyt.di;

import android.content.SharedPreferences;
import dagger.internal.Factory;
import javax.annotation.Generated;
import javax.inject.Provider;

@Generated(
    value = "dagger.internal.codegen.ComponentProcessor",
    comments = "https://dagger.dev"
)
@SuppressWarnings({
    "unchecked",
    "rawtypes"
})
public final class AppModule_ProvideFirstTimeToggleFactory implements Factory<Boolean> {
  private final Provider<SharedPreferences> sharedPrefProvider;

  public AppModule_ProvideFirstTimeToggleFactory(Provider<SharedPreferences> sharedPrefProvider) {
    this.sharedPrefProvider = sharedPrefProvider;
  }

  @Override
  public Boolean get() {
    return provideFirstTimeToggle(sharedPrefProvider.get());
  }

  public static AppModule_ProvideFirstTimeToggleFactory create(
      Provider<SharedPreferences> sharedPrefProvider) {
    return new AppModule_ProvideFirstTimeToggleFactory(sharedPrefProvider);
  }

  public static boolean provideFirstTimeToggle(SharedPreferences sharedPref) {
    return AppModule.INSTANCE.provideFirstTimeToggle(sharedPref);
  }
}
```

AppModule_ProvideNameFactory

```
package com.androiddevs.runningappyt.di;

import android.content.SharedPreferences;
import dagger.internal.Factory;
import dagger.internal.Preconditions;
import javax.annotation.Generated;
import javax.inject.Provider;

@Generated(
    value = "dagger.internal.codegen.ComponentProcessor",
    comments = "https://dagger.dev"
```

```
)
@SuppressWarnings({
    "unchecked",
    "rawtypes"
})
public final class AppModule_ProvideNameFactory implements Factory<String> {
  private final Provider<SharedPreferences> sharedPrefProvider;

  public AppModule_ProvideNameFactory(Provider<SharedPreferences> sharedPrefProvider) {
    this.sharedPrefProvider = sharedPrefProvider;
  }

  @Override
  public String get() {
    return provideName(sharedPrefProvider.get());
  }

  public static AppModule_ProvideNameFactory create(
      Provider<SharedPreferences> sharedPrefProvider) {
    return new AppModule_ProvideNameFactory(sharedPrefProvider);
  }

  public static String provideName(SharedPreferences sharedPref) {
    return Preconditions.checkNotNull(AppModule.INSTANCE.provideName(sharedPref), "Cannot return
null from a non-@Nullable @Provides method");
  }
}
```

AppModule_ProvideRunDaoFactory

```
package com.androiddevs.runningappyt.di;

import com.androiddevs.runningappyt.db.RunDAO;
import com.androiddevs.runningappyt.db.RunningDatabase;
import dagger.internal.Factory;
import dagger.internal.Preconditions;
import javax.annotation.Generated;
import javax.inject.Provider;

@Generated(
    value = "dagger.internal.codegen.ComponentProcessor",
    comments = "https://dagger.dev"
)
@SuppressWarnings({
    "unchecked",
    "rawtypes"
})
public final class AppModule_ProvideRunDaoFactory implements Factory<RunDAO> {
  private final Provider<RunningDatabase> dbProvider;

  public AppModule_ProvideRunDaoFactory(Provider<RunningDatabase> dbProvider) {
    this.dbProvider = dbProvider;
  }

  @Override
  public RunDAO get() {
    return provideRunDao(dbProvider.get());
  }

  public static AppModule_ProvideRunDaoFactory create(Provider<RunningDatabase> dbProvider) {
    return new AppModule_ProvideRunDaoFactory(dbProvider);
  }

  public static RunDAO provideRunDao(RunningDatabase db) {
```

```
    return Preconditions.checkNotNull(AppModule.INSTANCE.provideRunDao(db), "Cannot return null from
a non-@Nullable @Provides method");
  }
}
```

AppModule_ ProvideRunningDatabaseFactory

```
package com.androiddevs.runningappyt.di;

import android.content.Context;
import com.androiddevs.runningappyt.db.RunningDatabase;
import dagger.internal.Factory;
import dagger.internal.Preconditions;
import javax.annotation.Generated;
import javax.inject.Provider;

@Generated(
    value = "dagger.internal.codegen.ComponentProcessor",
    comments = "https://dagger.dev"
)
@SuppressWarnings({
    "unchecked",
    "rawtypes"
})
public final class AppModule_ProvideRunningDataBaseFactory implements Factory<RunningDatabase> {
  private final Provider<Context> appProvider;

  public AppModule_ProvideRunningDataBaseFactory(Provider<Context> appProvider) {
    this.appProvider = appProvider;
  }

  @Override
  public RunningDatabase get() {
    return provideRunningDataBase(appProvider.get());
  }

  public static AppModule_ProvideRunningDataBaseFactory create(Provider<Context> appProvider) {
    return new AppModule_ProvideRunningDataBaseFactory(appProvider);
  }

  public static RunningDatabase provideRunningDataBase(Context app) {
    return Preconditions.checkNotNull(AppModule.INSTANCE.provideRunningDataBase(app), "Cannot return
null from a non-@Nullable @Provides method");
  }
}
```

AppModule_ ProvideSharedPreferencesFactory

```
package com.androiddevs.runningappyt.di;

import android.content.Context;
import android.content.SharedPreferences;
import dagger.internal.Factory;
import dagger.internal.Preconditions;
import javax.annotation.Generated;
import javax.inject.Provider;

@Generated(
    value = "dagger.internal.codegen.ComponentProcessor",
    comments = "https://dagger.dev"
)
@SuppressWarnings({
    "unchecked",
    "rawtypes"
```

```
})
public final class AppModule_ProvideSharedPreferencesFactory implements Factory<SharedPreferences> {
  private final Provider<Context> appProvider;

  public AppModule_ProvideSharedPreferencesFactory(Provider<Context> appProvider) {
    this.appProvider = appProvider;
  }

  @Override
  public SharedPreferences get() {
    return provideSharedPreferences(appProvider.get());
  }

  public static AppModule_ProvideSharedPreferencesFactory create(Provider<Context> appProvider) {
    return new AppModule_ProvideSharedPreferencesFactory(appProvider);
  }

  public static SharedPreferences provideSharedPreferences(Context app) {
    return Preconditions.checkNotNull(AppModule.INSTANCE.provideSharedPreferences(app), "Cannot
return null from a non-@Nullable @Provides method");
  }
}
```

AppModule_ ProvideWeightFactory

```
package com.androiddevs.runningappyt.di;

import android.content.SharedPreferences;
import dagger.internal.Factory;
import javax.annotation.Generated;
import javax.inject.Provider;

@Generated(
    value = "dagger.internal.codegen.ComponentProcessor",
    comments = "https://dagger.dev"
)
@SuppressWarnings({
    "unchecked",
    "rawtypes"
})
public final class AppModule_ProvideWeightFactory implements Factory<Float> {
  private final Provider<SharedPreferences> sharedPrefProvider;

  public AppModule_ProvideWeightFactory(Provider<SharedPreferences> sharedPrefProvider) {
    this.sharedPrefProvider = sharedPrefProvider;
  }

  @Override
  public Float get() {
    return provideWeight(sharedPrefProvider.get());
  }

  public static AppModule_ProvideWeightFactory create(
      Provider<SharedPreferences> sharedPrefProvider) {
    return new AppModule_ProvideWeightFactory(sharedPrefProvider);
  }

  public static float provideWeight(SharedPreferences sharedPref) {
    return AppModule.INSTANCE.provideWeight(sharedPref);
  }
}
```

ServiceModule_ProvideBaseNotificationBuilderFactory

```
package com.androiddevs.runningappyt.di;

import android.app.PendingIntent;
import android.content.Context;
import androidx.core.app.NotificationCompat;
import dagger.internal.Factory;
import dagger.internal.Preconditions;
import javax.annotation.Generated;
import javax.inject.Provider;

@Generated(
    value = "dagger.internal.codegen.ComponentProcessor",
    comments = "https://dagger.dev"
)
@SuppressWarnings({
    "unchecked",
    "rawtypes"
})
public final class ServiceModule_ProvideBaseNotificationBuilderFactory implements
Factory<NotificationCompat.Builder> {
  private final Provider<Context> appProvider;

  private final Provider<PendingIntent> pendingIntentProvider;

  public ServiceModule_ProvideBaseNotificationBuilderFactory(Provider<Context> appProvider,
      Provider<PendingIntent> pendingIntentProvider) {
    this.appProvider = appProvider;
    this.pendingIntentProvider = pendingIntentProvider;
  }

  @Override
  public NotificationCompat.Builder get() {
    return provideBaseNotificationBuilder(appProvider.get(), pendingIntentProvider.get());
  }

  public static ServiceModule_ProvideBaseNotificationBuilderFactory create(
      Provider<Context> appProvider, Provider<PendingIntent> pendingIntentProvider) {
    return new ServiceModule_ProvideBaseNotificationBuilderFactory(appProvider,
pendingIntentProvider);
  }

  public static NotificationCompat.Builder provideBaseNotificationBuilder(Context app,
      PendingIntent pendingIntent) {
    return Preconditions.checkNotNull(ServiceModule.INSTANCE.provideBaseNotificationBuilder(app,
pendingIntent), "Cannot return null from a non-@Nullable @Provides method");
  }
}
```

ServiceModule_ProvideFusedLocationProviderClientFactory

```
package com.androiddevs.runningappyt.di;

import android.content.Context;
import com.google.android.gms.location.FusedLocationProviderClient;
import dagger.internal.Factory;
import dagger.internal.Preconditions;
import javax.annotation.Generated;
import javax.inject.Provider;

@Generated(
    value = "dagger.internal.codegen.ComponentProcessor",
    comments = "https://dagger.dev"
)
```

```
@SuppressWarnings({
    "unchecked",
    "rawtypes"
})
public final class ServiceModule_ProvideFusedLocationProviderClientFactory implements
Factory<FusedLocationProviderClient> {
  private final Provider<Context> appProvider;

  public ServiceModule_ProvideFusedLocationProviderClientFactory(Provider<Context> appProvider) {
    this.appProvider = appProvider;
  }

  @Override
  public FusedLocationProviderClient get() {
    return provideFusedLocationProviderClient(appProvider.get());
  }

  public static ServiceModule_ProvideFusedLocationProviderClientFactory create(
      Provider<Context> appProvider) {
    return new ServiceModule_ProvideFusedLocationProviderClientFactory(appProvider);
  }

  public static FusedLocationProviderClient provideFusedLocationProviderClient(Context app) {
    return
Preconditions.checkNotNull(ServiceModule.INSTANCE.provideFusedLocationProviderClient(app), "Cannot
return null from a non-@Nullable @Provides method");
  }
}
```

ServiceModule_ProvidemainActivityPendingIntentFactory

```
package com.androiddevs.runningappyt.di;

import android.app.PendingIntent;
import android.content.Context;
import dagger.internal.Factory;
import dagger.internal.Preconditions;
import javax.annotation.Generated;
import javax.inject.Provider;

@Generated(
    value = "dagger.internal.codegen.ComponentProcessor",
    comments = "https://dagger.dev"
)
@SuppressWarnings({
    "unchecked",
    "rawtypes"
})
public final class ServiceModule_ProvideMainActivityPendingIntentFactory implements
Factory<PendingIntent> {
  private final Provider<Context> appProvider;

  public ServiceModule_ProvideMainActivityPendingIntentFactory(Provider<Context> appProvider) {
    this.appProvider = appProvider;
  }

  @Override
  public PendingIntent get() {
    return provideMainActivityPendingIntent(appProvider.get());
  }

  public static ServiceModule_ProvideMainActivityPendingIntentFactory create(
      Provider<Context> appProvider) {
    return new ServiceModule_ProvideMainActivityPendingIntentFactory(appProvider);
```

```
  }

  public static PendingIntent provideMainActivityPendingIntent(Context app) {
    return Preconditions.checkNotNull(ServiceModule.INSTANCE.provideMainActivityPendingIntent(app),
"Cannot return null from a non-@Nullable @Provides method");
  }
}
```

MainRepository_Factory

```
package com.androiddevs.runningappyt.repositories;

import com.androiddevs.runningappyt.db.RunDAO;
import dagger.internal.Factory;
import javax.annotation.Generated;
import javax.inject.Provider;

@Generated(
    value = "dagger.internal.codegen.ComponentProcessor",
    comments = "https://dagger.dev"
)
@SuppressWarnings({
    "unchecked",
    "rawtypes"
})
public final class MainRepository_Factory implements Factory<MainRepository> {
  private final Provider<RunDAO> runDAOProvider;

  public MainRepository_Factory(Provider<RunDAO> runDAOProvider) {
    this.runDAOProvider = runDAOProvider;
  }

  @Override
  public MainRepository get() {
    return newInstance(runDAOProvider.get());
  }

  public static MainRepository_Factory create(Provider<RunDAO> runDAOProvider) {
    return new MainRepository_Factory(runDAOProvider);
  }

  public static MainRepository newInstance(RunDAO runDAO) {
    return new MainRepository(runDAO);
  }
}
```

Hilt_TrackingService

```
package com.androiddevs.runningappyt.services;

import androidx.annotation.CallSuper;
import androidx.lifecycle.LifecycleService;
import dagger.hilt.android.internal.managers.ServiceComponentManager;
import dagger.hilt.internal.GeneratedComponentManager;
import dagger.hilt.internal.UnsafeCasts;
import java.lang.Object;
import java.lang.Override;
import javax.annotation.Generated;

/**
 * A generated base class to be extended by the @dagger.hilt.android.AndroidEntryPoint annotated
class. If using the Gradle plugin, this is swapped as the base class via bytecode transformation.
 */
@Generated("dagger.hilt.android.processor.internal.androidentrypoint.ServiceGenerator")
```

```
public abstract class Hilt_TrackingService extends LifecycleService implements
GeneratedComponentManager<Object> {
  private volatile ServiceComponentManager componentManager;

  private final Object componentManagerLock = new Object();

  Hilt_TrackingService() {
    super();
  }

  @CallSuper
  @Override
  public void onCreate() {
    inject();
    super.onCreate();
  }

  protected ServiceComponentManager createComponentManager() {
    return new ServiceComponentManager(this);
  }

  protected final ServiceComponentManager componentManager() {
    if (componentManager == null) {
      synchronized (componentManagerLock) {
        if (componentManager == null) {
          componentManager = createComponentManager();
        }
      }
    }
    return componentManager;
  }

  protected void inject() {
    ((TrackingService_GeneratedInjector)
generatedComponent()).injectTrackingService(UnsafeCasts.<TrackingService>unsafeCast(this));
  }

  @Override
  public final Object generatedComponent() {
    return componentManager().generatedComponent();
  }
}
```

TrackingService_GeneratedInjector

```
package com.androiddevs.runningappyt.services;

import dagger.hilt.InstallIn;
import dagger.hilt.android.components.ServiceComponent;
import dagger.hilt.codegen.OriginatingElement;
import dagger.hilt.internal.GeneratedEntryPoint;
import javax.annotation.Generated;

@OriginatingElement(
    topLevelClass = TrackingService.class
)
@GeneratedEntryPoint
@InstallIn(ServiceComponent.class)
@Generated("dagger.hilt.android.processor.internal.androidentrypoint.InjectorEntryPointGenerator")
public interface TrackingService_GeneratedInjector {
  void injectTrackingService(TrackingService trackingService);
}
```

TrackingService_MembersInjector

```
package com.androiddevs.runningappyt.services;

import androidx.core.app.NotificationCompat;
import com.google.android.gms.location.FusedLocationProviderClient;
import dagger.MembersInjector;
import dagger.internal.InjectedFieldSignature;
import javax.annotation.Generated;
import javax.inject.Provider;

@Generated(
    value = "dagger.internal.codegen.ComponentProcessor",
    comments = "https://dagger.dev"
)
@SuppressWarnings({
    "unchecked",
    "rawtypes"
})
public final class TrackingService_MembersInjector implements MembersInjector<TrackingService> {
  private final Provider<FusedLocationProviderClient> fusedLocationProviderClientProvider;

  private final Provider<NotificationCompat.Builder> baseNotificationBuilderProvider;

  public TrackingService_MembersInjector(
      Provider<FusedLocationProviderClient> fusedLocationProviderClientProvider,
      Provider<NotificationCompat.Builder> baseNotificationBuilderProvider) {
    this.fusedLocationProviderClientProvider = fusedLocationProviderClientProvider;
    this.baseNotificationBuilderProvider = baseNotificationBuilderProvider;
  }

  public static MembersInjector<TrackingService> create(
      Provider<FusedLocationProviderClient> fusedLocationProviderClientProvider,
      Provider<NotificationCompat.Builder> baseNotificationBuilderProvider) {
    return new TrackingService_MembersInjector(fusedLocationProviderClientProvider,
baseNotificationBuilderProvider);
  }

  @Override
  public void injectMembers(TrackingService instance) {
    injectFusedLocationProviderClient(instance, fusedLocationProviderClientProvider.get());
    injectBaseNotificationBuilder(instance, baseNotificationBuilderProvider.get());
  }


@InjectedFieldSignature("com.androiddevs.runningappyt.services.TrackingService.fusedLocationProvider
Client")
  public static void injectFusedLocationProviderClient(TrackingService instance,
      FusedLocationProviderClient fusedLocationProviderClient) {
    instance.fusedLocationProviderClient = fusedLocationProviderClient;
  }


@InjectedFieldSignature("com.androiddevs.runningappyt.services.TrackingService.baseNotificationBuild
er")
  public static void injectBaseNotificationBuilder(TrackingService instance,
      NotificationCompat.Builder baseNotificationBuilder) {
    instance.baseNotificationBuilder = baseNotificationBuilder;
  }
}
```

BaseApplication_GeneratedInjector

```
package com.androiddevs.runningappyt;

import dagger.hilt.InstallIn;
import dagger.hilt.android.components.ApplicationComponent;
import dagger.hilt.codegen.OriginatingElement;
import dagger.hilt.internal.GeneratedEntryPoint;
import javax.annotation.Generated;

@OriginatingElement(
    topLevelClass = BaseApplication.class
)
@GeneratedEntryPoint
@InstallIn(ApplicationComponent.class)
@Generated("dagger.hilt.android.processor.internal.androidentrypoint.InjectorEntryPointGenerator")
public interface BaseApplication_GeneratedInjector {
  void injectBaseApplication(BaseApplication baseApplication);
}
```

BaseApplication_HiltComponents

```
package com.androiddevs.runningappyt;

import androidx.hilt.lifecycle.ViewModelFactoryModules;
import com.androiddevs.runningappyt.di.AppModule;
import com.androiddevs.runningappyt.di.ServiceModule;
import com.androiddevs.runningappyt.services.TrackingService_GeneratedInjector;
import com.androiddevs.runningappyt.ui.MainActivity_GeneratedInjector;
import com.androiddevs.runningappyt.ui.fragments.RunFragment_GeneratedInjector;
import com.androiddevs.runningappyt.ui.fragments.SettingsFragment_GeneratedInjector;
import com.androiddevs.runningappyt.ui.fragments.SetupFragment_GeneratedInjector;
import com.androiddevs.runningappyt.ui.fragments.StatisticsFragment_GeneratedInjector;
import com.androiddevs.runningappyt.ui.fragments.TrackingFragment_GeneratedInjector;
import com.androiddevs.runningappyt.ui.viewmodels.MainViewModel_HiltModule;
import com.androiddevs.runningappyt.ui.viewmodels.StatisticsViewModel_HiltModule;
import dagger.Binds;
import dagger.Component;
import dagger.Module;
import dagger.Subcomponent;
import dagger.hilt.android.components.ActivityComponent;
import dagger.hilt.android.components.ActivityRetainedComponent;
import dagger.hilt.android.components.ApplicationComponent;
import dagger.hilt.android.components.FragmentComponent;
import dagger.hilt.android.components.ServiceComponent;
import dagger.hilt.android.components.ViewComponent;
import dagger.hilt.android.components.ViewWithFragmentComponent;
import dagger.hilt.android.internal.builders.ActivityComponentBuilder;
import dagger.hilt.android.internal.builders.ActivityRetainedComponentBuilder;
import dagger.hilt.android.internal.builders.FragmentComponentBuilder;
import dagger.hilt.android.internal.builders.ServiceComponentBuilder;
import dagger.hilt.android.internal.builders.ViewComponentBuilder;
import dagger.hilt.android.internal.builders.ViewWithFragmentComponentBuilder;
import dagger.hilt.android.internal.lifecycle.DefaultViewModelFactories;
import dagger.hilt.android.internal.managers.ActivityComponentManager;
import dagger.hilt.android.internal.managers.FragmentComponentManager;
import
dagger.hilt.android.internal.managers.HiltWrapper_ActivityRetainedComponentManager_LifecycleComponen
tBuilderEntryPoint;
import dagger.hilt.android.internal.managers.ServiceComponentManager;
import dagger.hilt.android.internal.managers.ViewComponentManager;
import dagger.hilt.android.internal.modules.ApplicationContextModule;
import dagger.hilt.android.internal.modules.HiltWrapper_ActivityModule;
import dagger.hilt.android.scopes.ActivityRetainedScoped;
import dagger.hilt.android.scopes.ActivityScoped;
```

```java
import dagger.hilt.android.scopes.FragmentScoped;
import dagger.hilt.android.scopes.ServiceScoped;
import dagger.hilt.android.scopes.ViewScoped;
import dagger.hilt.internal.GeneratedComponent;
import dagger.hilt.migration.DisableInstallInCheck;
import javax.annotation.Generated;
import javax.inject.Singleton;

@Generated("dagger.hilt.processor.internal.root.RootProcessor")
public final class BaseApplication_HiltComponents {
  private BaseApplication_HiltComponents() {
  }

  @Module(
      subcomponents = ActivityC.class
  )
  @DisableInstallInCheck
  @Generated("dagger.hilt.processor.internal.root.RootProcessor")
  abstract interface ActivityCBuilderModule {
    @Binds
    ActivityComponentBuilder bind(ActivityC.Builder builder);
  }

  @Module(
      subcomponents = ActivityRetainedC.class
  )
  @DisableInstallInCheck
  @Generated("dagger.hilt.processor.internal.root.RootProcessor")
  abstract interface ActivityRetainedCBuilderModule {
    @Binds
    ActivityRetainedComponentBuilder bind(ActivityRetainedC.Builder builder);
  }

  @Module(
      subcomponents = FragmentC.class
  )
  @DisableInstallInCheck
  @Generated("dagger.hilt.processor.internal.root.RootProcessor")
  abstract interface FragmentCBuilderModule {
    @Binds
    FragmentComponentBuilder bind(FragmentC.Builder builder);
  }

  @Module(
      subcomponents = ServiceC.class
  )
  @DisableInstallInCheck
  @Generated("dagger.hilt.processor.internal.root.RootProcessor")
  abstract interface ServiceCBuilderModule {
    @Binds
    ServiceComponentBuilder bind(ServiceC.Builder builder);
  }

  @Module(
      subcomponents = ViewC.class
  )
  @DisableInstallInCheck
  @Generated("dagger.hilt.processor.internal.root.RootProcessor")
  abstract interface ViewCBuilderModule {
    @Binds
    ViewComponentBuilder bind(ViewC.Builder builder);
  }
```

```
@Module(
    subcomponents = ViewWithFragmentC.class
)
@DisableInstallInCheck
@Generated("dagger.hilt.processor.internal.root.RootProcessor")
abstract interface ViewWithFragmentCBuilderModule {
  @Binds
  ViewWithFragmentComponentBuilder bind(ViewWithFragmentC.Builder builder);
}

@Subcomponent(
    modules = {
        FragmentCBuilderModule.class,
        ViewCBuilderModule.class,
        DefaultViewModelFactories.ActivityModule.class,
        HiltWrapper_ActivityModule.class,
        ViewModelFactoryModules.ActivityModule.class
    }
)
@ActivityScoped
public abstract static class ActivityC implements MainActivity_GeneratedInjector,
    ActivityComponent,
    DefaultViewModelFactories.ActivityEntryPoint,
    FragmentComponentManager.FragmentComponentBuilderEntryPoint,
    ViewComponentManager.ViewComponentBuilderEntryPoint,
    GeneratedComponent {
  @Subcomponent.Builder
  abstract interface Builder extends ActivityComponentBuilder {
  }
}

@Subcomponent(
    modules = {
        ActivityCBuilderModule.class,
        MainViewModel_HiltModule.class,
        StatisticsViewModel_HiltModule.class
    }
)
@ActivityRetainedScoped
public abstract static class ActivityRetainedC implements ActivityRetainedComponent,
    ActivityComponentManager.ActivityComponentBuilderEntryPoint,
    GeneratedComponent {
  @Subcomponent.Builder
  abstract interface Builder extends ActivityRetainedComponentBuilder {
  }
}

@Component(
    modules = {
        AppModule.class,
        ApplicationContextModule.class,
        ActivityRetainedCBuilderModule.class,
        ServiceCBuilderModule.class
    }
)
@Singleton
public abstract static class ApplicationC implements BaseApplication_GeneratedInjector,
    ApplicationComponent,
    HiltWrapper_ActivityRetainedComponentManager_LifecycleComponentBuilderEntryPoint,
    ServiceComponentManager.ServiceComponentBuilderEntryPoint,
    GeneratedComponent {
}
```

```
  @Subcomponent(
      modules = {
          ViewWithFragmentCBuilderModule.class,
          DefaultViewModelFactories.FragmentModule.class,
          ViewModelFactoryModules.FragmentModule.class
      }
  )
  @FragmentScoped
  public abstract static class FragmentC implements RunFragment_GeneratedInjector,
      SettingsFragment_GeneratedInjector,
      SetupFragment_GeneratedInjector,
      StatisticsFragment_GeneratedInjector,
      TrackingFragment_GeneratedInjector,
      FragmentComponent,
      DefaultViewModelFactories.FragmentEntryPoint,
      ViewComponentManager.ViewWithFragmentComponentBuilderEntryPoint,
      GeneratedComponent {
    @Subcomponent.Builder
    abstract interface Builder extends FragmentComponentBuilder {
    }
  }

  @Subcomponent(
      modules = ServiceModule.class
  )
  @ServiceScoped
  public abstract static class ServiceC implements TrackingService_GeneratedInjector,
      ServiceComponent,
      GeneratedComponent {
    @Subcomponent.Builder
    abstract interface Builder extends ServiceComponentBuilder {
    }
  }

  @Subcomponent
  @ViewScoped
  public abstract static class ViewC implements ViewComponent,
      GeneratedComponent {
    @Subcomponent.Builder
    abstract interface Builder extends ViewComponentBuilder {
    }
  }

  @Subcomponent
  @ViewScoped
  public abstract static class ViewWithFragmentC implements ViewWithFragmentComponent,
      GeneratedComponent {
    @Subcomponent.Builder
    abstract interface Builder extends ViewWithFragmentComponentBuilder {
    }
  }
}
```

Hilt_BaseApplication

```
package com.androiddevs.runningappyt;

import android.app.Application;
import androidx.annotation.CallSuper;
import dagger.hilt.android.internal.managers.ApplicationComponentManager;
import dagger.hilt.android.internal.managers.ComponentSupplier;
import dagger.hilt.android.internal.modules.ApplicationContextModule;
import dagger.hilt.internal.GeneratedComponentManager;
import dagger.hilt.internal.UnsafeCasts;
```

```
import java.lang.Object;
import java.lang.Override;
import javax.annotation.Generated;

/**
 * A generated base class to be extended by the @dagger.hilt.android.HiltAndroidApp annotated class.
If using the Gradle plugin, this is swapped as the base class via bytecode transformation.
 */
@Generated("dagger.hilt.android.processor.internal.androidentrypoint.ApplicationGenerator")
public abstract class Hilt_BaseApplication extends Application implements
GeneratedComponentManager<Object> {
  private final ApplicationComponentManager componentManager = new ApplicationComponentManager(new
ComponentSupplier() {
    @Override
    public Object get() {
      return DaggerBaseApplication_HiltComponents_ApplicationC.builder()
          .applicationContextModule(new ApplicationContextModule(Hilt_BaseApplication.this))
          .build();
    }
  });

  protected final ApplicationComponentManager componentManager() {
    return componentManager;
  }

  @Override
  public final Object generatedComponent() {
    return componentManager().generatedComponent();
  }

  @CallSuper
  @Override
  public void onCreate() {
    // This is a known unsafe cast, but is safe in the only correct use case:
    // BaseApplication extends Hilt_BaseApplication
    ((BaseApplication_GeneratedInjector)
generatedComponent()).injectBaseApplication(UnsafeCasts.<BaseApplication>unsafeCast(this));
    super.onCreate();
  }
}
```

RunFragmentsDirections

```
package com.androiddevs.runningappyt.ui.fragments

import androidx.navigation.ActionOnlyNavDirections
import androidx.navigation.NavDirections
import com.androiddevs.runningappyt.NavGraphDirections
import com.androiddevs.runningappyt.R

class RunFragmentDirections private constructor() {
  companion object {
    fun actionRunFragmentToTrackingFragment(): NavDirections =
        ActionOnlyNavDirections(R.id.action_runFragment_to_trackingFragment)

    fun actionGlobalTrackingFragment(): NavDirections =
        NavGraphDirections.actionGlobalTrackingFragment()
  }
}
```

SettingsFragmentsDirections

```
package com.androiddevs.runningappyt.ui.fragments
```

```
import androidx.navigation.NavDirections
import com.androiddevs.runningappyt.NavGraphDirections

class SettingsFragmentDirections private constructor() {
  companion object {
    fun actionGlobalTrackingFragment(): NavDirections =
        NavGraphDirections.actionGlobalTrackingFragment()
  }
}
```

SetupFragmentsDirections

```
package com.androiddevs.runningappyt.ui.fragments

import androidx.navigation.ActionOnlyNavDirections
import androidx.navigation.NavDirections
import com.androiddevs.runningappyt.NavGraphDirections
import com.androiddevs.runningappyt.R

class SetupFragmentDirections private constructor() {
  companion object {
    fun actionSetupFragmentToRunFragment(): NavDirections =
        ActionOnlyNavDirections(R.id.action_setupFragment_to_runFragment)

    fun actionGlobalTrackingFragment(): NavDirections =
        NavGraphDirections.actionGlobalTrackingFragment()
  }
}
```

StatisticsFragmentsDirections

```
package com.androiddevs.runningappyt.ui.fragments

import androidx.navigation.NavDirections
import com.androiddevs.runningappyt.NavGraphDirections

class StatisticsFragmentDirections private constructor() {
  companion object {
    fun actionGlobalTrackingFragment(): NavDirections =
        NavGraphDirections.actionGlobalTrackingFragment()
  }
}
```

TrackingFragmentsDirections

```
package com.androiddevs.runningappyt.ui.fragments

import androidx.navigation.ActionOnlyNavDirections
import androidx.navigation.NavDirections
import com.androiddevs.runningappyt.NavGraphDirections
import com.androiddevs.runningappyt.R

class TrackingFragmentDirections private constructor() {
  companion object {
    fun actionTrackingFragmentToRunFragment(): NavDirections =
        ActionOnlyNavDirections(R.id.action_trackingFragment_to_runFragment)

    fun actionGlobalTrackingFragment(): NavDirections =
        NavGraphDirections.actionGlobalTrackingFragment()
  }
}
```

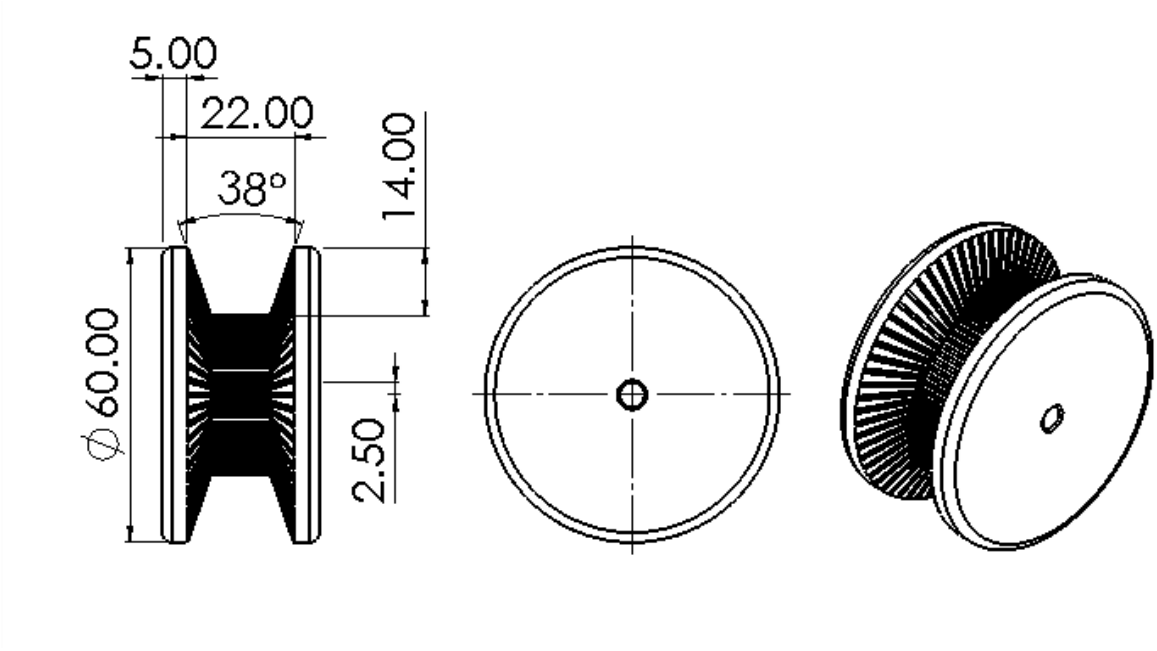
NavGraphDirections

```
package com.androiddevs.runningappyt

import androidx.navigation.ActionOnlyNavDirections
import androidx.navigation.NavDirections

class NavGraphDirections private constructor() {
  companion object {
    fun actionGlobalTrackingFragment(): NavDirections =
        ActionOnlyNavDirections(R.id.action_global_tracking_fragment)
  }
}
```

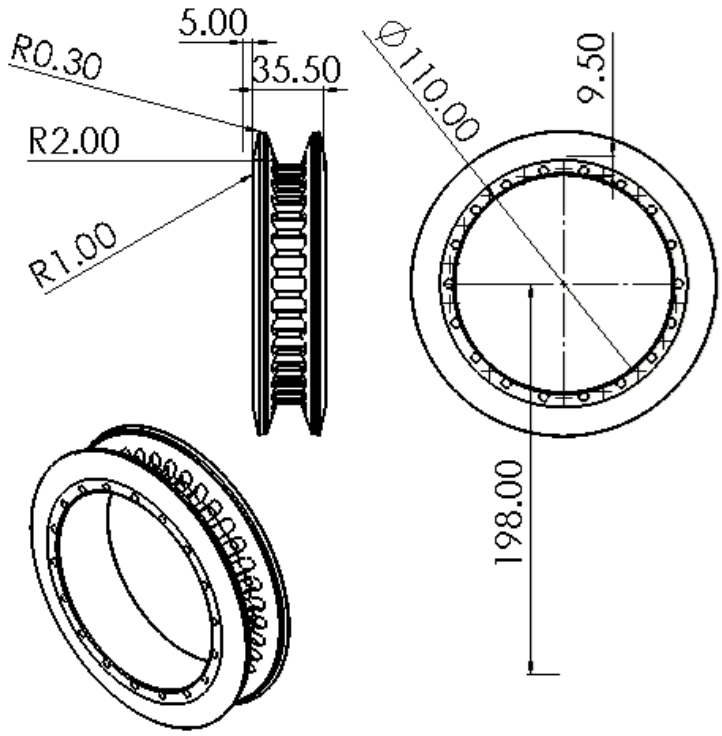
Anexo 3

Planos de las piezas a manufacturar

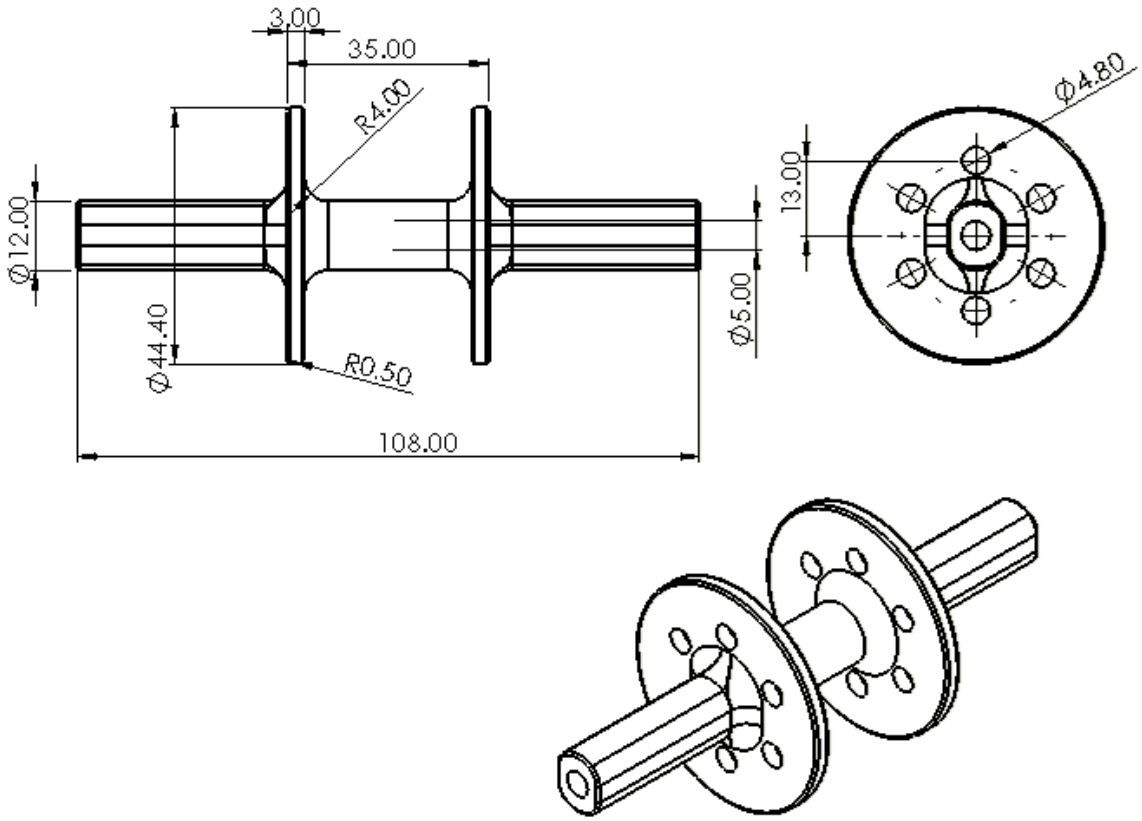
Rueda lateral tipo polea



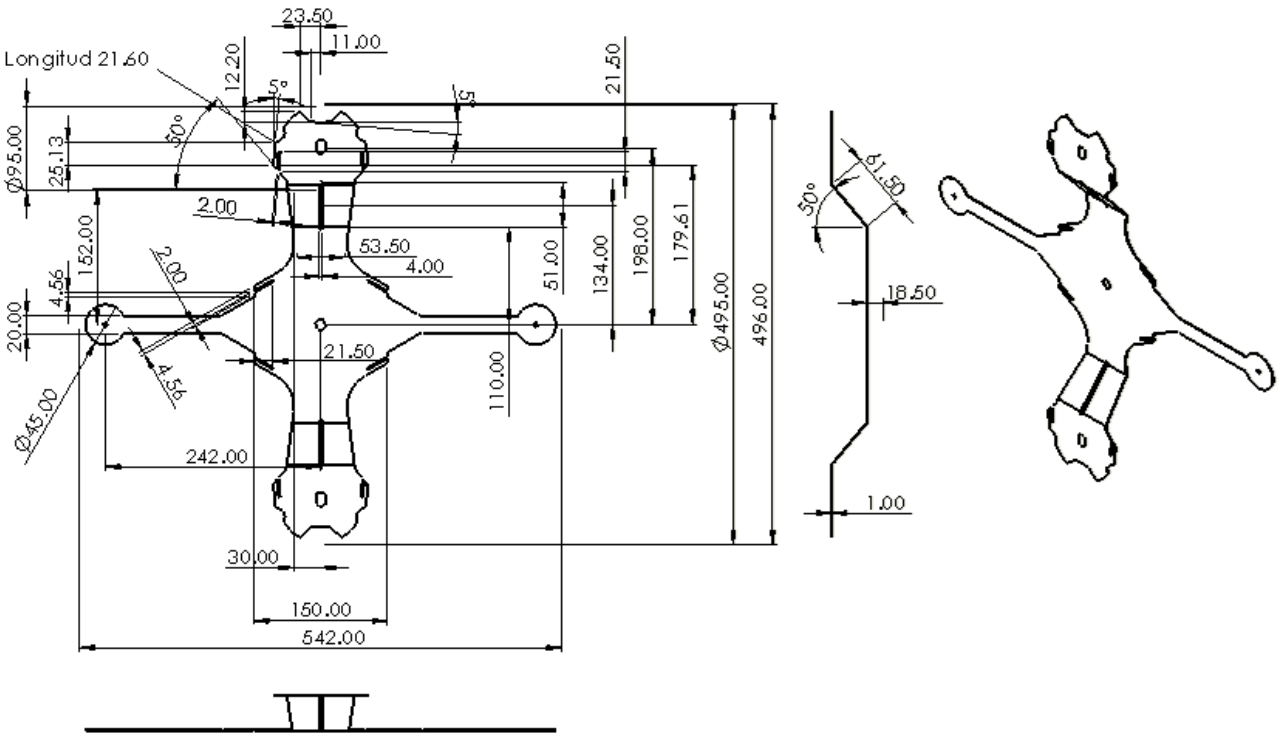
Acoplamiento mecánico motor MXUS-XF07 – Banda trapezoidal C64



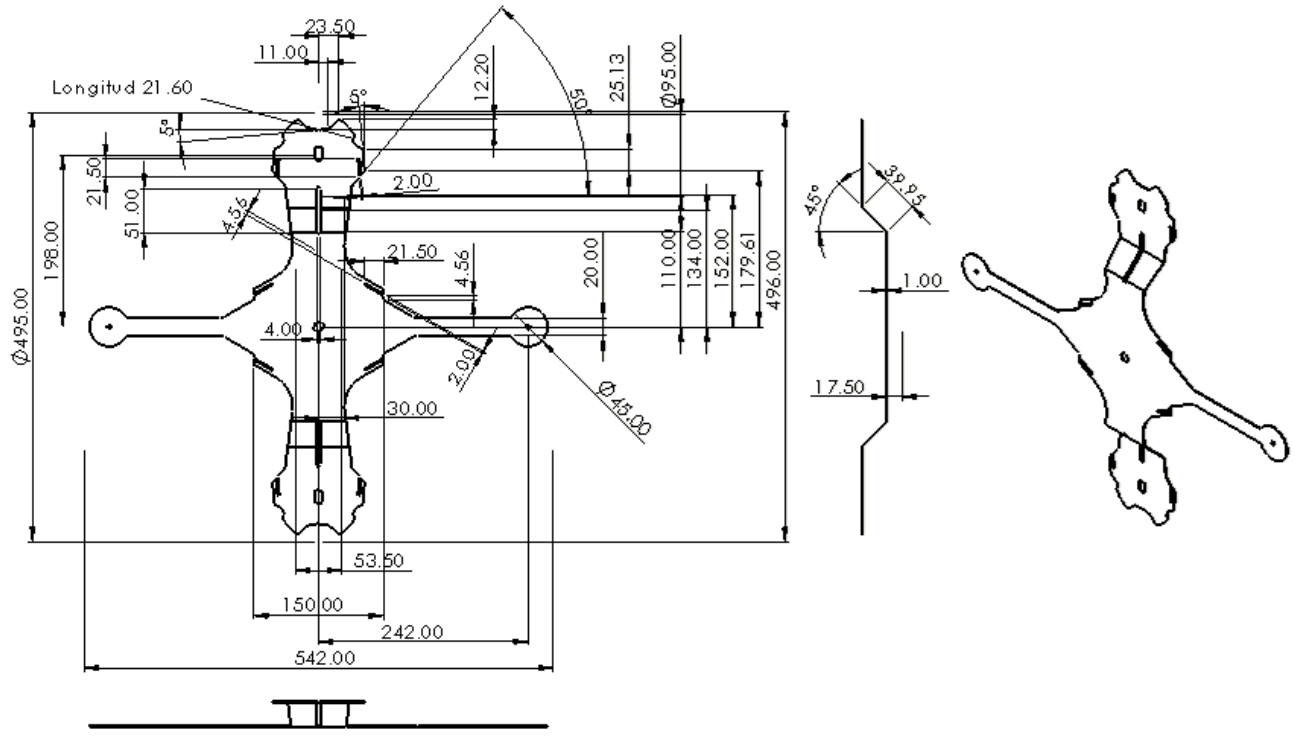
Eje de rotación para el eje de cierre rápido



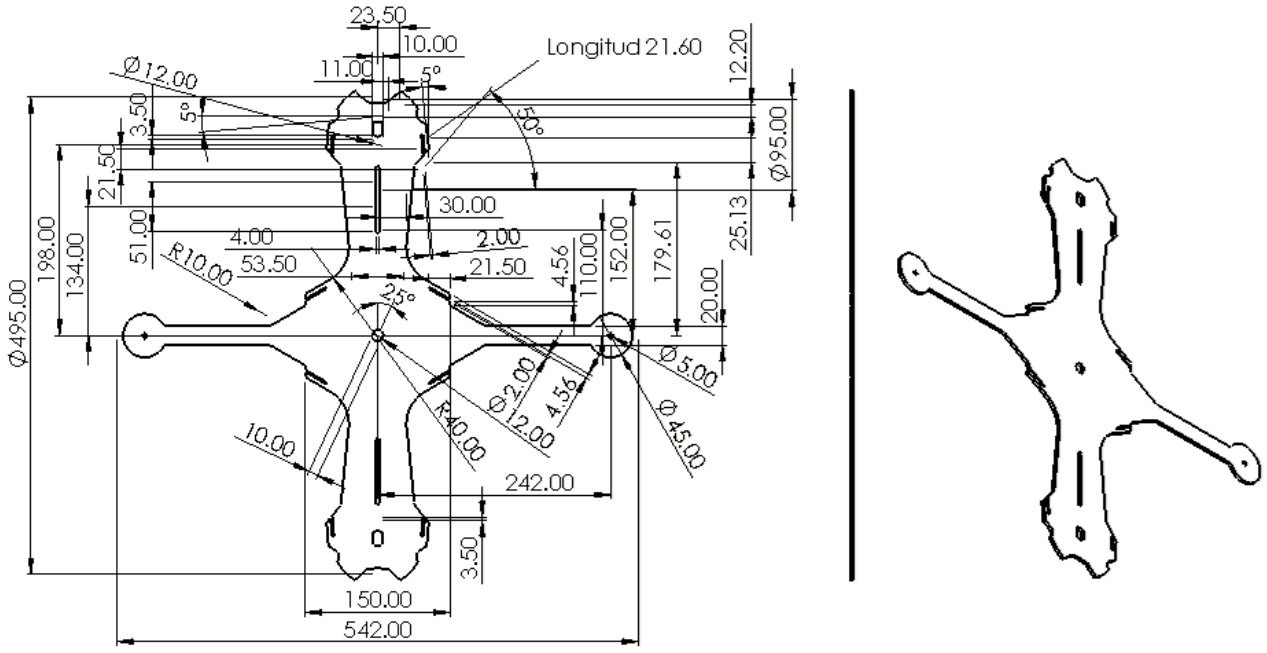
Placa lateral derecha estructura mecánica - Medidas



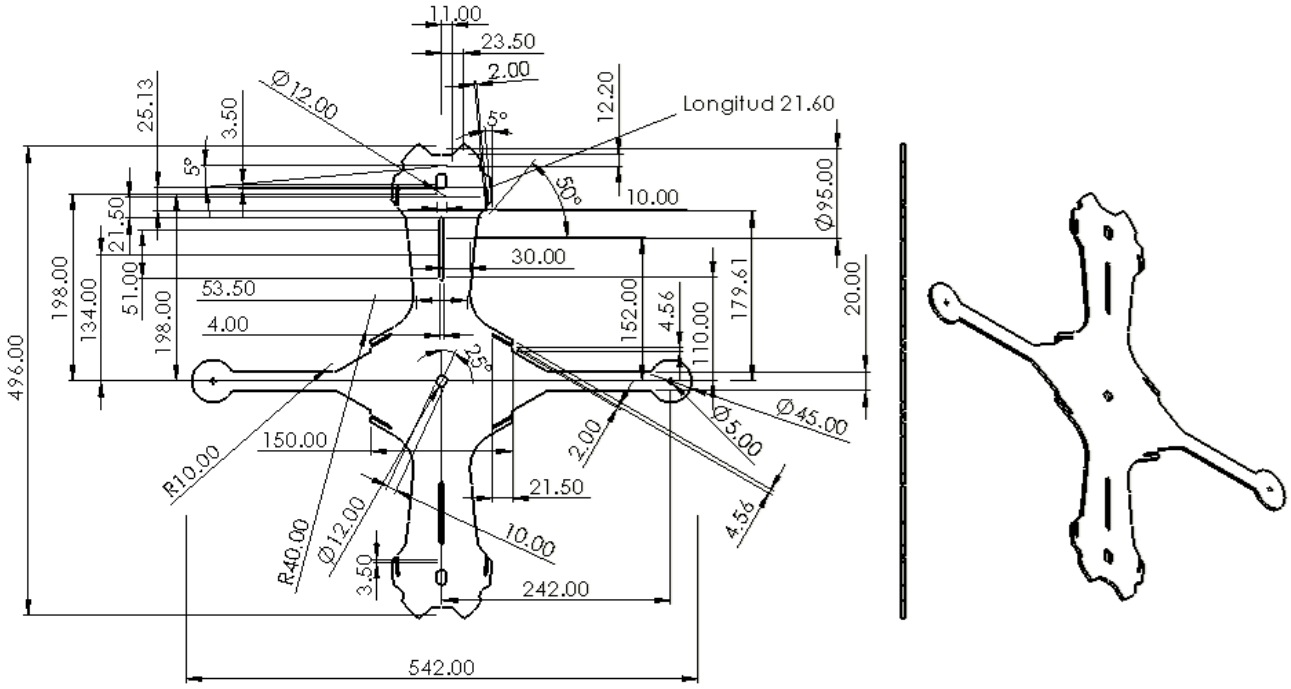
Placa lateral izquierda estructura mecánica - Medidas



Placa lateral derecha estructura mecánica – Plano para el CNC

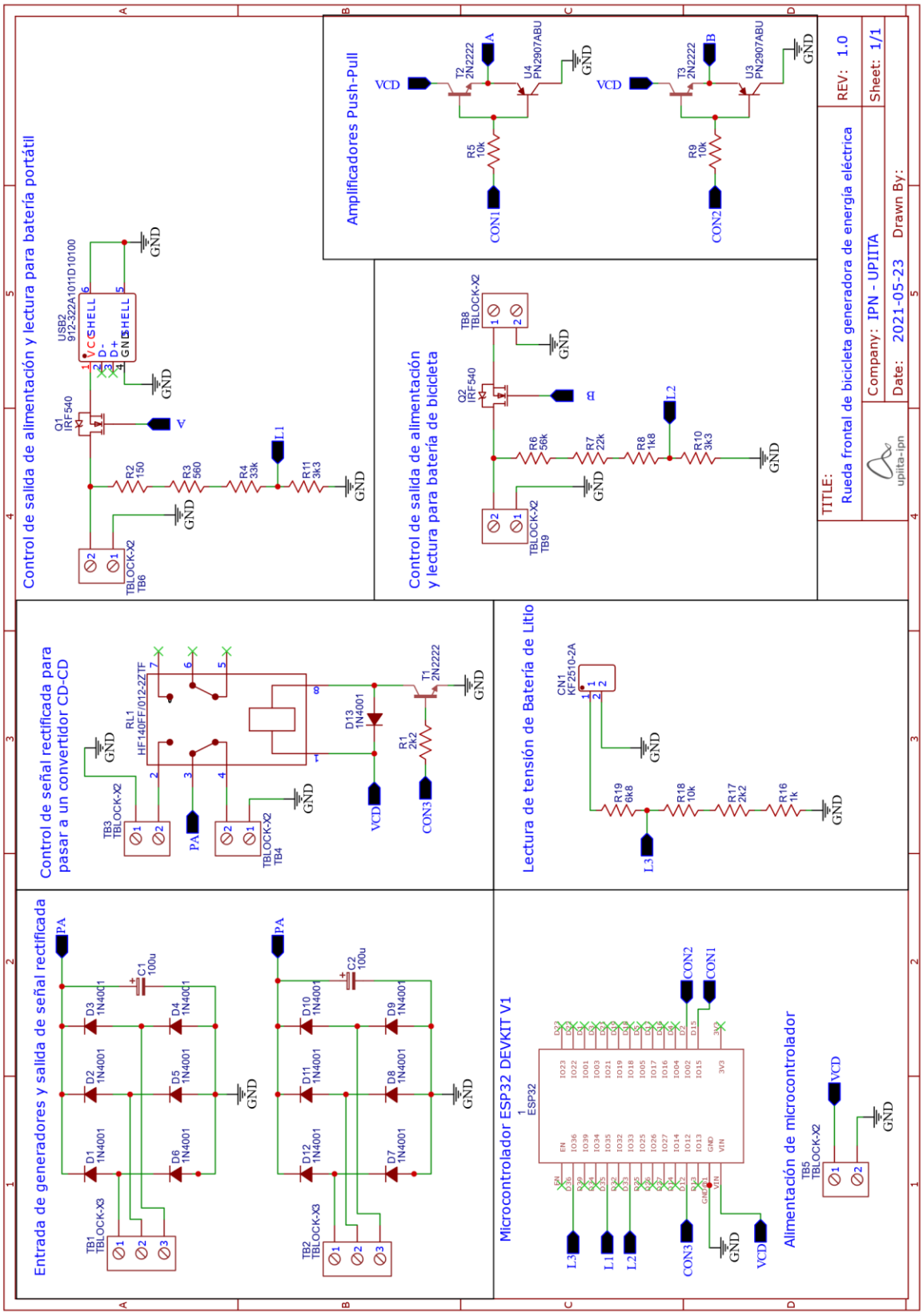


Placa lateral izquierda estructura mecánica – Plano para el CNC de plasma



Anexo 4

Circuito general de rueda frontal de bicicleta generadora de energía eléctrica



Anexo 5

Tabla 51. Anexo 5. Selección de lámina de aluminio considerando Fmax, Amin y el Factor de seguridad de cada calibre y aleación.

Fy max = 473.9739 N ancho mínimo = 57 mm = 0.057 m			Factor de seguridad						
			Aluminio 6061-T6, T651 (AISI-ATM)	Aluminio 2014-T6, T651 (AISI-ATM)	Aluminio 2024-T861 (AISI-ATM)	Aluminio 5056-H18 (AISI-ATM)	Aluminio 2618-T61 (AISI-ATM)	Aluminio 3003-H14 (AISI-ATM)	Aluminio 1100-H14 (AISI-ATM)
			240.0000	269.0000	310.0000	283.0000	262.0000	82.7371	69.0000
calibre	σt (MPa)	espesor (metros)	F.S.1	F.S.2	F.S.3	F.S.4	F.S.5	F.S.6	F.S.7
4	1.3744	0.00605	174.6172	195.7168	225.5472	205.9028	190.6238	60.1972	50.2024
6	1.6115	0.00516	148.9297	166.9254	192.3676	175.6130	162.5816	51.3417	42.8173
8	1.9846	0.00419	120.9332	135.5460	156.2054	142.6004	132.0188	41.6903	34.7683
10	2.4457	0.0034	98.1320	109.9896	126.7538	115.7140	107.1274	33.8298	28.2129
12	3.0019	0.00277	79.9487	89.6092	103.2671	94.2728	87.2773	27.5614	22.9853
14	3.9409	0.00211	60.8996	68.2583	78.6619	71.8107	66.4820	20.9944	17.5086
16	5.0396	0.00165	47.6229	53.3773	61.5129	56.1553	51.9883	16.4174	13.6916
18	6.7059	0.00124	35.7893	40.1139	46.2279	42.2016	39.0700	12.3379	10.2894
19	8.2330	0.00101	29.1510	32.6734	37.6533	34.3739	31.8231	10.0494	8.3809
20	9.3431	0.00089	25.6875	28.7914	33.1797	30.2898	28.0422	8.8555	7.3852
22	11.6953	0.000711	20.5211	23.0008	26.5065	24.1978	22.4022	7.0744	5.8998
24	14.8754	0.000559	16.1341	18.0836	20.8398	19.0247	17.6130	5.5620	4.6385
26	18.1955	0.000457	13.1901	14.7839	17.0372	15.5533	14.3992	4.5471	3.7922

Fy max = 473.9739 N ancho mínimo = 57 mm = 0.057 m			Aluminio 6061-T6, T651 (AISI-ATM)
			240.0000
calibre	σt (MPa)	espesor (metros)	F.S.1
4	1.3744	0.00605	174.6172
6	1.6115	0.00516	148.9297
8	1.9846	0.00419	120.9332
10	2.4457	0.0034	98.1320
12	3.0019	0.00277	79.9487
14	3.9409	0.00211	60.8996
16	5.0396	0.00165	47.6229
18	6.7059	0.00124	35.7893
19	8.2330	0.00101	29.1510
20	9.3431	0.00089	25.6875
22	11.6953	0.000711	20.5211
24	14.8754	0.000559	16.1341
26	18.1955	0.000457	13.1901

Aluminio 2014-T6, T651 (AISI-ATM)	Aluminio 2024-T861 (AISI-ATM)
269.0000	310.0000
F.S.2	F.S.3
195.7168	225.5472
166.9254	192.3676
135.5460	156.2054
109.9896	126.7538
89.6092	103.2671
68.2583	78.6619
53.3773	61.5129
40.1139	46.2279
32.6734	37.6533
28.7914	33.1797
23.0008	26.5065
18.0836	20.8398
14.7839	17.0372

Factor de seguridad

Aluminio 5056-H18 (AISI-ATM)	Aluminio 2618-T61 (AISI-ATM)
283.0000	262.0000
F.S.4	**F.S.5**
205.9028	190.6238
175.6130	162.5816
142.6004	132.0188
115.7140	107.1274
94.2728	87.2773
71.8107	66.4820
56.1553	51.9883
42.2016	39.0700
34.3739	31.8231
30.2898	28.0422
24.1978	22.4022
19.0247	17.6130
15.5533	14.3992

Aluminio 3003-H14 (AISI-ATM)	Aluminio 1100-H14 (AISI-ATM)
82.7371	69.0000
F.S.6	**F.S.7**
60.1972	50.2024
51.3417	42.8173
41.6903	34.7683
33.8298	28.2129
27.5614	22.9853
20.9944	17.5086
16.4174	13.6916
12.3379	10.2894
10.0494	8.3809
8.8555	7.3852
7.0744	5.8998
5.5620	4.6385
4.5471	3.7922

Anexo 6

Señales trifásicas del motor brushless.

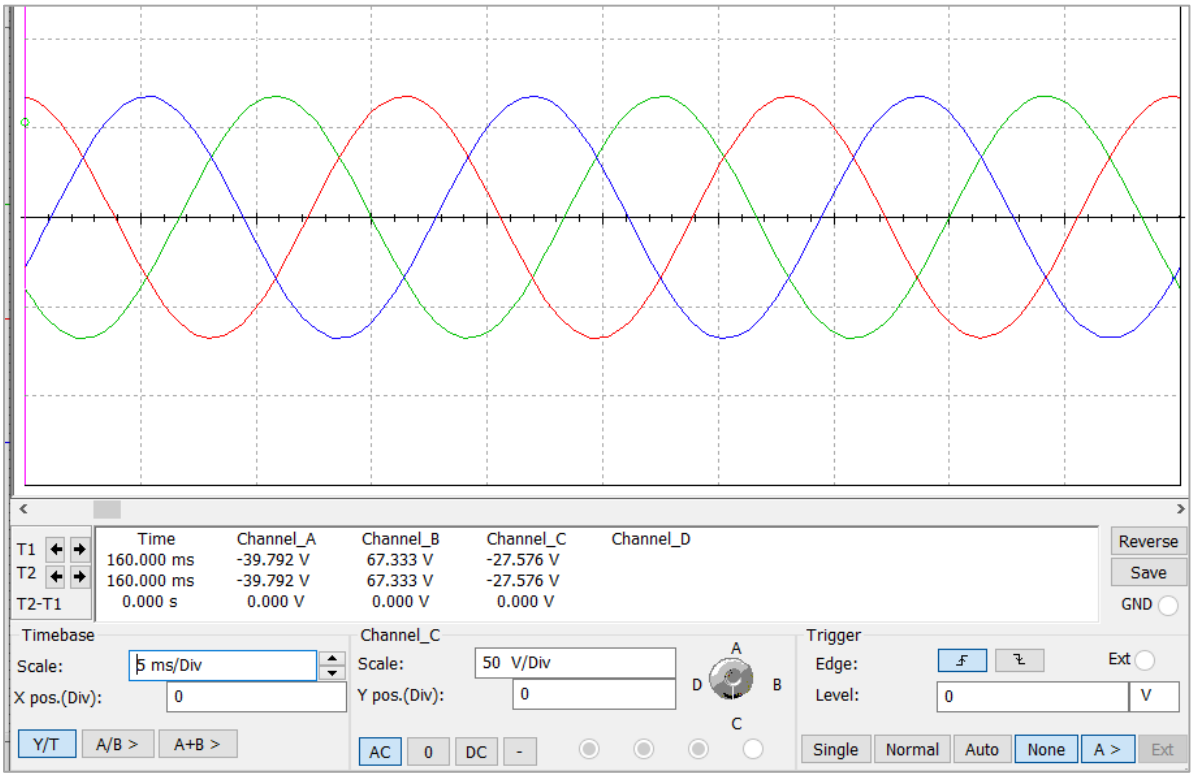


Figura 163. Anexo 6. Señal trifásica proveniente de las terminales del motor sin escobillas.

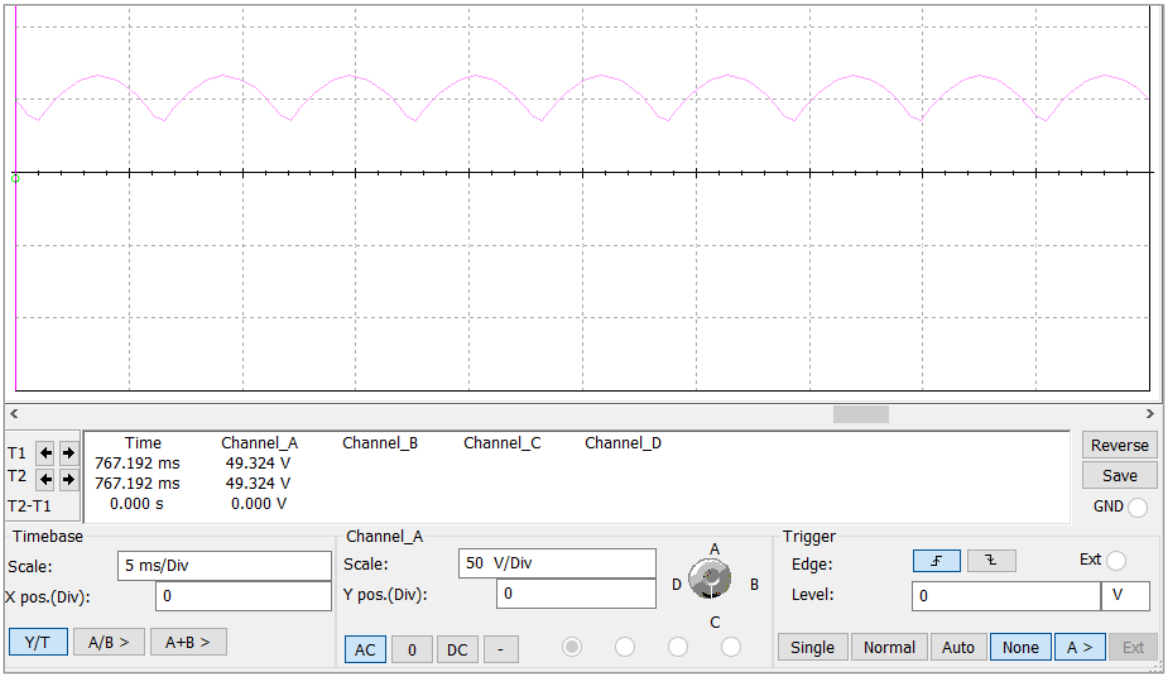


Figura 164. Anexo 6. Señal trifásica rectificada.

Anexo 7

Gráfica característica de saturación del MOSFET IRF-540N

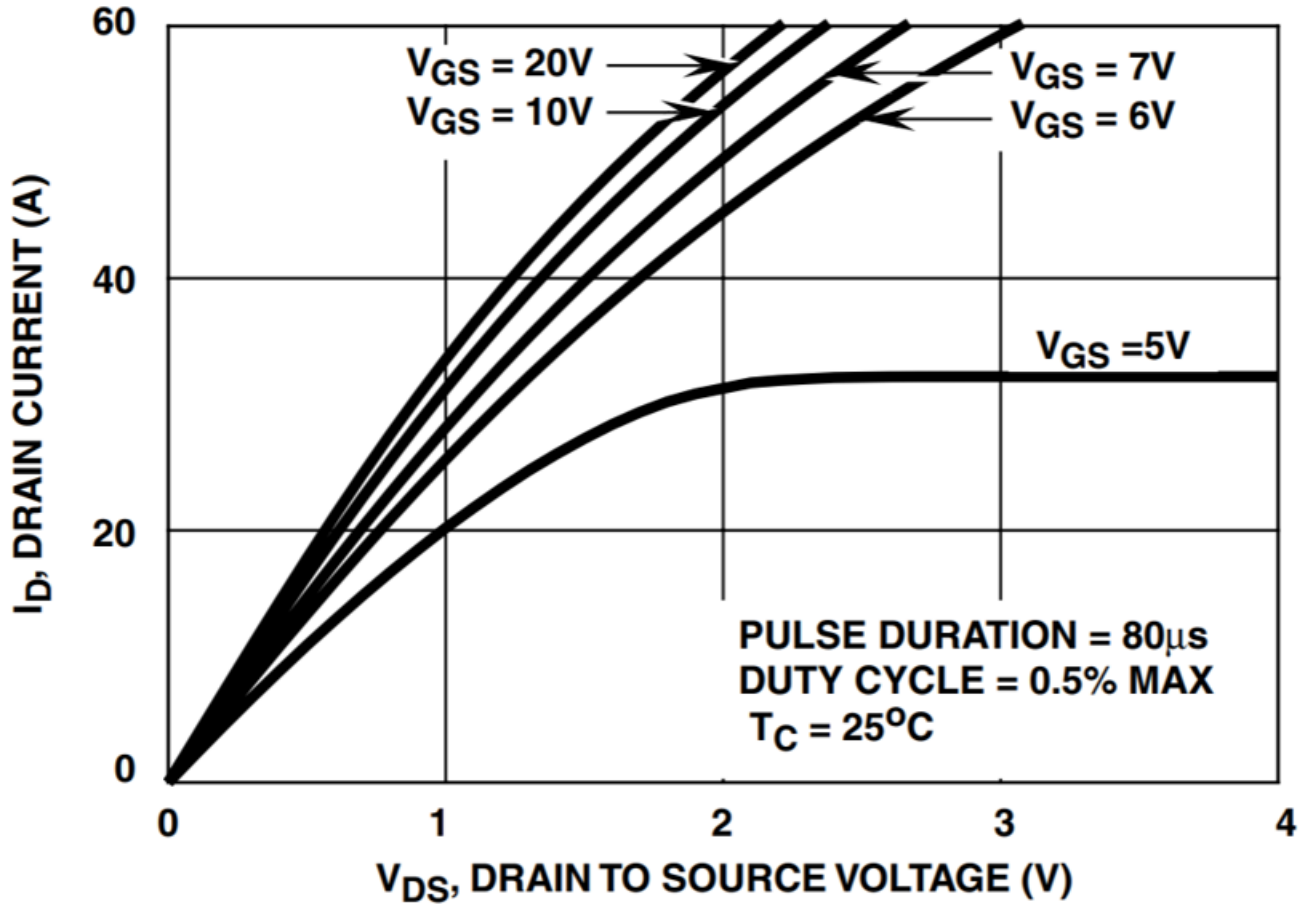


FIGURE 8. SATURATION CHARACTERISTICS