DISEÑO CONCEPTUAL, PRELIMINAR Y DETALLADO DEL COHETE SONDA RECUPERABLE “ARISTARCO I” PROPULSADO CON PROPELENTE SOLIDO.

OSCAR DAVID SALAZAR CEBALLOS

MARIO ALEJANDRO SOLORZANO GOMEZ

FUNDACION UNIVERSITARIA LOS LIBERTADORES
FACULTAD DE INGENIERIAS
INGENIERIA AERONAUTICA
BOGOTA, D.C.
2016

DISEÑO CONCEPTUAL, PRELIMINAR Y DETALLADO DEL COHETE SONDA RECUPERABLE “ARISTARCO I” PROPULSADO CON PROPELENTE SOLIDO.

OSCAR DAVID SALAZAR CEBALLOS

MARIO ALEJANDRO SOLORZANO GOMEZ

Trabajo de grado para optar por el título de Ingeniero Aeronáutico

Director

Diego Armando Reyes Caballero
Ingeniero Aeronáutico

FUNDACION UNIVERSITARIA LOS LIBERTADORES
FACULTAD DE INGENIERIAS
INGENIERIA AERONAUTICA
BOGOTA, D.C.
2016

Nota de aceptación

_____

_____

_____

_____

_____

_____

_____
Firma de presidente del jurado

_____
Firma del jurado

_____
Firma del jurado

Bogotá, D. C. 31 de agosto 2016

Las directivas de la Fundación Universitaria Los Libertadores, los jurados calificadores y el cuerpo docente no son responsable por los criterios e ideas expuestas en el presente documento. Estos corresponden únicamente a los autores

Dedicado a mi familia en especial a mi tío Fabián, a mis compañeros e ingenieros que, con su ayuda y determinación, me permitieron culminar este sueño. Gracias.

Oscar David Salazar Ceballos

Dedicado a mi familia, por su apoyo a lo largo de la carrera, a mis amigos y compañeros que brindaron su apoyo y confianza durante la formación profesional y el desarrollo de este proyecto.

Mario Alejandro Solórzano Gómez

AGRADECIMIENTOS

Los autores expresan su agradecimiento a:

Ingeniero Diego Reyes Caballero, por el apoyo, confianza, interés y acompañamiento mostrados durante el proceso del desarrollo del presente documento, desde el inicio hasta su culminación, mediante el aporte constante de su conocimiento y experiencia al proyecto.

Próximo Ingeniero Electrónico Eduardo Borrero Cruz, por brindar acompañamiento y guía en los aspectos electrónicos que fueron desarrollados para el vehículo.

Próximas Ingenieras Aeronáuticas Gerliz Amado Morales y Lina Mendoza Mendoza, por la generosidad con que brindaron apoyo, a partir de su trabajo de grado en el desarrollo de un sistema esencial para el vehículo diseñado en este documento, como lo es el sistema de recuperación.

Coordinación de laboratorios, salas y talleres de la Fundación Universitaria Los Libertadores, por la colaboración prestada en la disposición de espacios para el desarrollo del proyecto.

TABLA DE CONTENIDO

pág.

LISTA DE TABLAS

pág.

LISTA DE IMÁGENES

pág.

LISTA DE ECUACIONES

pág.

LISTA DE GRAFICAS

pág.

LISTA DE DIAGRAMAS

pág.

LISTA DE ANEXOS

pág.

LISTA DE SIMBOLOS Y ABREVIATURAS

$\overline{X_F}$	Centro Geométrico del Fuselaje
$\overline{X_O}$	Centro Geométrico de la Ojiva
$\overline{X_{PL}}$	Centro geométrico de la carga paga
$\overline{X_{SP}}$	Centro geométrico del sistema de propulsión
$\overline{X_{SR}}$	Centro geométrico del sistema de recuperación
$Ø_{Conv}$	Diámetro de entrada de la Tobera
$Ø_{Div}$	Diámetro de salida de la Tobera
$Ø_G$	Diámetro de la garganta de la Tobera
$Ø_S$	Diámetro de salida de la tobera
$Ø_f$	Diámetro del fuselaje
$Ø_{fe}$	Diámetro exterior del fuselaje
$Ø_{fi}$	Diámetro interior del fuselaje
$Ø_o$	Diámetro de la Ojiva
A_A	Superficie de la Aleta Sustentadora
A_F	Área expuesta del Fuselaje
A_O	Área expuesta de la Ojiva
A_Q	Área de quemado de la cámara de combustión
A_{TO}	Área expuesta de la Tobera
A_{TOT}	Área Total expuesta
A_b	Superficie de la base del cuerpo
A_{latF}	Área Lateral del Fuselaje
A_s	Área de salida de la tobera
C^*	Velocidad de descarga característica
C_{Root}	Cuerda de Raíz
C_{Tip}	Cuerda de Punta
C_e	Coeficiente de empuje
I_{SP}	Impulso Especifico
I_{Tot}	Impulso total
M_S	Numero mach de salida de la tobera
M_f	Masa Final del Cohete
M_i	Masa Inicial del Cohete
P_{C-Ad}	Presión máxima admisible en la cámara de combustión
P_{C-T}	Presión de trabajo en la cámara de combustión
P_e	Presión exterior
R'	Constante universal de los gases
S_p	Superficie del paracaídas
S_u	Superficie
T_C	Temperatura ideal de combustión
T_M	Empuje del motor
V_A	Volumen de una Aleta Sustentadora
V_{AisPL}	Volumen del Aislante de la Carga Paga
V_{Des}	Velocidad de descenso del cohete
V_{Max}	Velocidad máxima del cohete
V_{Prop}	Volumen del propelente
V_R	Volumen Real

V_S	Velocidad de salida de la tobera
V_{aisc}	Volumen del aislante de la cámara de combustión
V_{cas}	Volumen de la cámara de combustión
V_{cil}	Volumen del Cilindro
V_e	Volumen Externo
V_f	Volumen del Fuselaje
V_i	Volumen Interno
V_o	Volumen de la Ojiva
W_p	Peso del propelente
W_t	Peso total del cohete
X_{AC}	Localización del centro aerodinámico
a_s	Velocidad de propagación del sonido
d_A	Distancia de LR a centro geométrico de las Aletas Estabilizadoras
d_{AisPL}	Distancia de LR a centro geométrico del aislante de la carga paga
d_{CG}	Distancia de LR a centro de gravedad
d_{CP}	Distancia de LR a centro de presiones
d_F	Distancia de LR a centro geométrico del Fuselaje
d_O	Distancia de LR a centro geométrico de la Ojiva
d_{PL}	Distancia de LR a centro geométrico de la carga paga
d_{SP}	Distancia de LR a centro geométrico del sistema de propulsión
d_{SR}	Distancia de LR a centro geométrico del sistema de recuperación
d_{TO}	Distancia de LR a centro geométrico de la Tobera
e_A	Espesor de la Aletas Estabilizadoras
e_o	Espesor de la Ojiva
h_{Max}	Altura máxima del cohete
h_{comb}	Altura que alcanza el cohete durante la combustión
l_{Cas}	Longitud del cámara de combustión
l_O	Longitud de la Ojiva
l_T	Longitud Total
l_f	Longitud del fuselaje
l_{pl}	Longitud de la Carga Paga
$\dot{m}$	Flujo másico
m_A	Masa de las Aletas Estabilizadoras
m_{AisC}	Masa del Aislante de la cámara de combustión
m_{AisPL}	Masa del Aislante de la Carga Paga
m_{AisPL}	Masa del aislante de la carga paga
m_F	Masa del Fuselaje
m_O	Masa de la Ojiva
m_P	Masa del propelente
m_{PL}	Masa de la carga paga
m_{SP}	Masa del sistema de propulsión
m_{SR}	Masa del sistema de recuperación
m_{Tot}	Masa total del cohete
r_C	Radio del cámara de combustión
r_{Cas}	Radio del cámara de combustión
r_G	Radio de la garganta de la tobera
r_S	Radio de salida de la tobera
t_{Tot}	Tiempo de ascenso del cohete

ρ_A	Densidad de la Aleta Sustentadora
ρ_{aisc}	Densidad del aislante de la cámara de combustión
ρ_{cas}	Densidad de la cámara de combustión
ρ_f	Densidad del Fuselaje
ρ_o	Densidad de la Ojiva
A	Aletas
AG	Apogeo
AisC	Aislante de la cámara de combustión
AisPL	Aislante de la Carga Paga
AP	Adaptación
AT	Aislante Térmico
Cas	Cámara de Combustión
Cd	Coeficiente de Drag
CG	Centro de Gravedad
CP	Centro de Presiones
D	Drag
E	Energía Eléctrica
ES	Estructura
ESA	Agencia Espacial Europea
F	Fuselaje
F	Fuselaje
g	Aceleración de la gravedad
h	Altura de un cuerpo prismático
-I	Componentes disponibles en el semillero Aerodes&i
K	Relación de calor específico del propelente
Kn	Klemmung
L	Lastre
LR	Línea de Referencia
M	Misión
m	Masa
M	Peso molecular efectivo
M	Misión
Mach	Numero Mach
ME	Margen de Estabilidad
ML	Masa de Lanzamiento
O	Ojiva
OT	Operación y Sistema en Tierra
P	Propelente
PL	Carga Paga
PUR	Espuma de Poliuretano
r	Radio
R	Constante del gas
RCS	Radar Cross Section
RM	Relación de Masa
RM	Relación de Masa
RP	Relación de presión
S	Borde de fuga
S	Borde de Fuga
SP	Sistema de Propulsión
SR	Sistema de Recuperación

t	Tiempo de quemado del propelente
T	Temperatura del aire
T	Motor
To	Tobera
TR	Instrumentos
V	Volumen
a	Coeficiente de combustión
e	Espesor
l	Longitud
n	Exponente de combustión
v	Velocidad
β	Angulo de la sección divergente de la Tobera
γ	Coeficiente de dilatación adiabática
θ	Angulo de la sección convergente de la Tobera
ρ	Densidad
σ	Limite elástico del material

INTRODUCCIÓN

En el presente documento se encuentra el desarrollo del diseño conceptual, preliminar y detallado de un cohete sonda propulsado por propelente sólido, llamado *ARISTARCO I,* cuya misión principal es la obtención de datos atmosféricos (presión y temperatura) por encima de un kilómetro de altitud; sin embargo, a partir de los sistemas desarrollados, el cohete a su vez puede realizar mediciones de altitud, tiempo, aceleración e inclinación del vehículo.

Inicialmente se conceptualizan los componentes del cohete, y a través de variables cualitativas, se determina el tipo de cada uno de los componentes que serán implementados en el vehículo.

Posteriormente se realiza el diseño preliminar, en el cual se dimensionan y evalúan los componentes que conforman el *ARISTARCO I.* De esta manera se conocen de forma cuantitativa, parámetros fundamentales en el desarrollo de la misión del cohete.

Por último, se obtienen los planos de cada uno de los componentes evaluados en las dos fases iníciales y del conjunto del vehículo diseñado, a partir de los cuales, se explica minuciosamente como van incorporados entre sí, dichos componentes para integrar un solo cohete sonda.

PLANTEAMIENTO DEL PROBLEMA

La implementación de tecnología obtenida en investigaciones desarrolladas en el marco aeroespacial, aplicadas al campo aeronáutico, ha ido aumentando progresivamente a través de los últimos años, debido a que los resultados de la inmensa investigación realizada en torno a la conquista del espacio, ha dejado como resultado un amplio número de avances tecnológicos esencialmente en nuevos materiales utilizados con mejores propiedades que los usados convencionalmente, componentes estructurales, diseño, métodos de construcción y características de vehículos, los cuales en función de su eficiencia, versatilidad y calidad, es posible incluirlos en el desarrollo de la satisfacción de necesidades que hoy en día presenta el sector aeronáutico.

En Colombia, hasta el momento, el campo aeroespacial no ha tenido desarrollo, debido a la falta de apoyo económico por parte del sector gubernamental. Dicho apoyo, es fundamental para que Colombia pueda iniciar su carrera aeroespacial y sea autónoma,competitiva en el campo en mención, con respecto a países que ya han desarrollado el campo aeroespacial.

Actualmente, con el fin de obtener información de condiciones atmosféricas, se realiza diariamente la adquisición de datos de la mencionada información, a través de *GLOBOS SONDA*, los que satisfacen la necesidad de obtener dicha información; sin embargo estos presentan una baja velocidad de ascenso, lo que implica que la obtención de los datos requeridos, conlleve un periodo de tiempo considerable, adicionalmente en la actualidad, los globos sonda no son recuperables, lo que produce un gasto económico elevado, debido a que al realizar cada medición, es necesario el uso de un nuevo equipo *globo sonda*.[1]

La adquisición de datos atmosféricos en Colombia, se ha caracterizado por realizar la medición de estos a través de la implementación de un dispositivo llamado *globo sonda,* el cual está compuesto por una sección de adquisición de datos, que consta de sensores para realizar la medición de la información mencionada, y los circuitos electrónicos para generar la transmisión de estos datos a la estación en tierra; la otra sección es el globo, el cual es el encargado de ascender el conjunto de las dos secciones mencionadas a una altitud determinada. Una vez el *globo sonda* llega a la altitud mencionada, en la que se terminan de tomar los datos requeridos, este estalla, y el conjunto del dispositivo se precipita a tierra, este dispositivo termina en un lugar desconocido y prácticamente destruido debido al impacto con la superficie terrestre. El *globo sonda* no posee un sistema de recuperación que permita la reutilización del dispositivo. Esto hace que para cada medición de datos que se realice, sea necesario utilizar un dispositivo completamente nuevo.

[1] NASA, Columbia Scientific Balloon Facility, Scientific Balloon; [en línea].

Realizar la toma de datos mediante un globo sonda, toma aproximadamente 30 minutos, a una altitud cercana a los 10km;[2] realizar dicha tarea, sería posible en un menor periodo de tiempo al que comprende actualmente, mediante un vehículo con propulsión de combustible, de ascenso vertical y que logre alcanzar la altitud requerida, ya que el tiempo total que consume el ascenso y descenso de un vehículo a la altitud ya mencionada, es de máximo 20 minutos.[3]

Teniendo en cuenta las razones anteriormente descritas, se produce la pregunta: ¿Qué características de diseño orientadas a un vehículo propulsado, permiten optimizar la adquisición de datos atmosféricos e incentiva el desarrollo aeroespacial en Colombia?

[2] Union Internacional de Comunicaciones. CARACTERÍSTICAS TÉCNICAS Y CRITERIOS DE CALIDAD DE LOS SISTEMAS DE RADIOSONDAS DEL SERVICIO DE AYUDAS A LA METEOROLOGÍA.

[3]Primer Cohete-sonda argentino de dos etapas, [En linea].

JUSTIFICACIÓN

Desarrollar un proyecto de este tipo, fomenta la rama de estudio de vehículos aeroespaciales en el área de *mecánica aeroespacial* el cual, ha sido incentivado por proyectos de investigación previos en el programa Ingeniería Aeronáutica de la Fundación Universitaria Los Libertadores, no obstante, este no ha tenido el desarrollo esperado, a pesar de contar con numerosos proyectos, los cuales han buscado fomentar la investigación aeroespacial;sin embargo, ninguno de estos ha sido desarrollado a cabalidad. Mediante el desarrollo de un cohete sonda recuperable, las distintas disciplinas que conforman la Ingeniería Aeronáutica pueden integrarse y ser aplicadas, en este caso al área aeroespacial.

El desarrollo del *"ARISTARCO I"*, responde a la necesidad de optimizar la recolección de datos tomados de condiciones atmosféricas, a una altitud superior a 1 Km, postulando una alternativa que haría el proceso de dicha adquisición más eficaz y eficiente; adicionalmente propicia el estudio y desarrollo de vehículos aeroespaciales en Colombia, que de lograr la construcción sería un proyecto de gran avance en esta área, mediante las bases que puedan permitir proyectos realizados, tanto los que han sido desarrollados dentro de la Fundación Universitaria Los Libertadores como en otras instituciones.

La adquisición de datos atmosféricos no es exclusivamente de utilidad para la operación de aeronaves, sino también para la información administrada por el *Instituto de Hidrología, Meteorología y Estudios Ambientalesde Colombia (IDEAM)*, los cuales son funcionales al ingresar en el registro de estadísticas del mencionado instituto; para poder ofrecer información a la población colombiana, en la que se hace una aproximación pronosticada de cómo será el comportamiento climático durante el día.[4] El *"ARISTARCO I"*proporciona de manera efectiva, acertada y en un periodo de tiempo muy breve la información en cuestión.

El área de estudio aeroespacial, actualmente es uno de los campos que más investigación está experimentando, el desarrollo de este, se encuentra en un proceso de crecimiento extremadamente veloz; el espacio es el futuro cuasi inmediato de la industria aeronáutica, y Colombia necesita obtener importancia en dicha industria; importancia y vigencia que perdió la oportunidad de tener, cuando quedo rezagado por una gran cantidad de países latinoamericanos en el desarrollo aeronáutico en la región, a pesar de ser pionero en el campoaeronáutico a nivel de Latinoamérica. La oportunidad de desarrollar un proyecto que promueva y origine el desarrollo aeroespacial colombiano, es de gran inspiración y motivación como actuales estudiantes y próximos ingenieros aeronáuticos

[4] Instituto de Hidrología, Meteorología y Estudios Ambientales – IDEAM. Resolución No. 085 de Mayo 4 de 2006

OBJETIVOS

OBJETIVO GENERAL

Realizar el diseño conceptual, preliminar y detallado del cohete *“ARISTARCO I”*recuperable, propulsado por combustible sólido, con fines de adquisición de datos de presión y temperatura atmosféricas.

OBJETIVOS ESPECIFICOS:

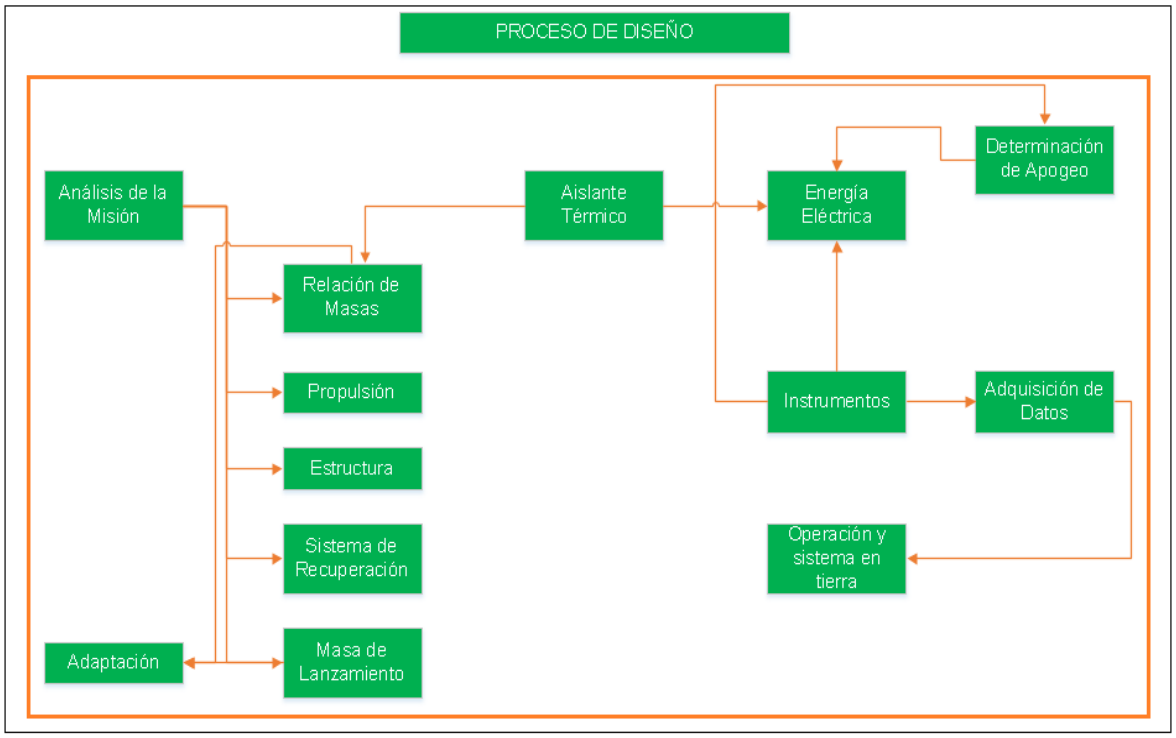
- Describir los principales sistemas que conforman cohetes sonda.
- Seleccionar el tipo de vehículo y los sistemas que conformaran el *“ARISTARCO I”*, teniendo en cuenta las condiciones de operación en las cuales se basara el diseño.
- Establecer las características y componentes de los sistemas de propulsión, recuperación, adquisición de datos y lanzamiento a implementar en el Cohete *“ARISTARCO I”*, teniendo en cuenta el tipo de vehículo y los sistemas seleccionados.
- Realizar el dimensionamiento y la disposición de cada uno de los componentes que integra el cohete sonda “*ARISTARCO I”.*

1. DISEÑO CONCEPTUAL

El diseño conceptual es desarrollado con una metodología basada en el sistema de diseño de vehículos espaciales de la Agencia Espacial Europea (ESA)[5], en el cual se expone una manera efectiva de obtener resultados verídicos y en concordancia con cada uno de los ítems que se tienen en cuenta para el diseño de un cohete.

A continuación,en el diagrama 1, se observa la metodología usada para el desarrollo del diseño del cohete sonda ARISTARCO I, mediante la cual, se evaluará cada uno de los ítems que pertenecen al "Proceso de diseño", en los cuales, cada uno de ellos, se relacionara con los demás ítems que se encuentran directamente afectados.Para una mayor facilidad de lectura del documento, se propone a cada ítem asignar determinada letra, para una posterior conexión en el diagrama final del diseño conceptual, las cuales se encuentran establecidas en la lista de símbolos y abreviaturas.

Diagrama 1. Metodología de Diseño del Cohete Sonda ARISTARCO I



Fuente: Agencia Espacial Europea - ESA

[5] M. Bandecchi & B. Melton, F. Ongaro; Concurrent Engineering Applied to Space Mission Assessment and Design.1999.

1.1. Misión

La utilización de los cohetes sonda tiene gran cantidad de propósitos, diversos entre sí, como el transporte de personal, equipos aeroespaciales, suministros, instrumentos de comunicación para puesta en órbita como satélites, dispositivos de medición de propiedades atmosféricas, a un punto del espacio, determinado por una altitud, que tiene como referencia el nivel del mar terrestre.[6]

Para el cohete sonda Aristarco I, se determina una misión de transporte de un dispositivo de medición de propiedades atmosféricas como presión, temperatura y adicional la altitud a la que el aire presenta las características medidas. El cual debe ser recuperable y reutilizable, que cuenta con un motor cohete de propelente sólido, y un alcance de altitud máximo (apogeo) por encima de 1Km.

1.2. Relación de Masa

El término de ingeniería aeroespacial, “Relación de Masa”, es utilizado para definir cuantitativamente la eficiencia másica de un vehículo determinado, es decir, que tanta masa tiene el vehículo con propelente, que sin este; esto es la relación existente entre la masa seca y masa húmeda del vehículo aeroespacial en cuestión. Donde la masa seca se define como la masa del vehículo y la masa de la carga paga de este, mientras que la masa húmeda, es sumatoria la masa del cohete con la masa de la carga paga y del propelente. Por lo anteriormente descrito, en el diseño de cohetes y vehículos espaciales, siempre se busca obtener un número alto de relación de masa (valor cercano a 1); ya que esto implica menor cantidad de combustible para llevar a cabo la misión para la que esté destinado dicho vehículo. La relación de masa (RM) está dada por la ecuación:[7]

Ecuación 1.Relación de Masa

$$RM = \frac{M_f}{M_i}$$

Fuente: Libro Rocket propulsión elements. p. 29.

Donde RM es la relación de masa descrita anteriormente; M_f es definida como la masa final, es decir la masa del cohete después de haber culminado la misión,

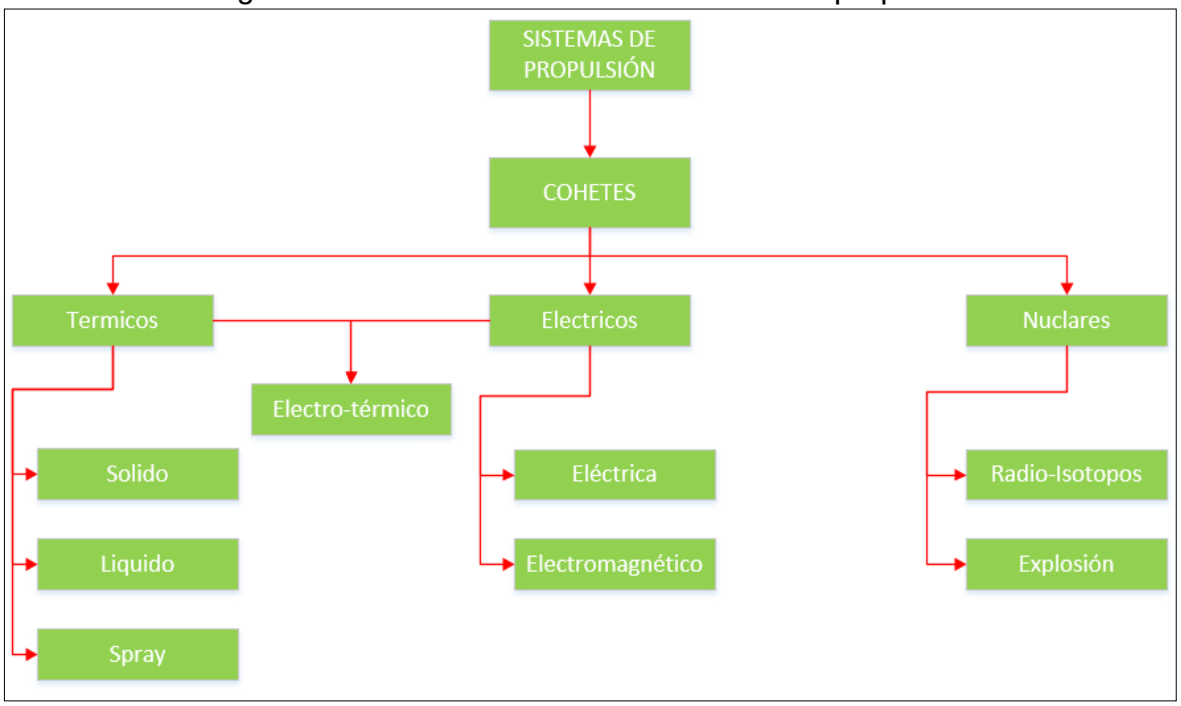
[6] Stark, Jhon; Swinerd, Graham y Fortescue Peter. Spacecraft Systems Engineering. Inglaterra: Wiley, 2003. p. 1 – 4.
[7] Libro Rocket Propulsion Elements. p. 29.

cuando el sistema propulsor ha consumido todo el propelente establecido; $M_i$es la masa que tiene el cohete antes de iniciar la misión, es decir la masa inicial del cohete contando el propelente suministrado.[8]

1.3. Sistema de Propulsión

El sistema de propulsión para vehículos espaciales se distingue en el Diagrama 2. De las varias alternativas identificadas, los dispositivos se han ido centrando actualmente en sistemas de propulsión térmicos y eléctricos[9].

Diagrama 2. Clasificación de los sistemas de propulsión.



Fuente: Autores

Es de vital importancia conocer cada una de las características de los diferentes sistemas de propulsión térmica y eléctrica para determinar el sistema de propulsión adecuado para el cohete sonda ARISTARCO I.

[8]Ibid. p. 29.
[9] Stark, Jhon; Swinerd, Graham y Fortescue Peter. Spacecraft Systems Engineering. Inglaterra: Wiley, 2003. p. 169 – 170.

- Sistemas de Propulsión Térmicos.
- Sistema de Propulsión Eléctricos

A diferencia de los sistemas de propulsión térmicos que se han descrito hasta el momento, los sistemas de propulsión eléctricos, requiere de una energía necesaria para la propulsión del cohete. Si esta fuente es la radiación solar o combustible nuclear, puede ser prácticamente limitada en cuanto al rendimiento y los niveles de empuje alcanzados; en los sistemas de propulsión eléctrica encontramos los siguientes:

- Sistema de Propulsión Eléctrica
- Sistema de Propulsión Electromagnético

Para el caso del sistema de propulsión del cohete sonda ARISTARCO I se ha optado el sistema de propulsión térmico; esto se da ya que la quema de propulsores químicos ya sea el propelente solido o líquido, a alta presión libera grandes cantidades de energía en un volumen compacto, convirtiendo la energía química en energía cinética generando una propulsión al cohete sonda[10].

1.4. Propelente

En principio, cualquier sistema químico que produce la liberación de calor en un flujo de gas a través de una reacción exotérmica podría ser utilizado como un propulsor. En la práctica, la consideración de una serie de factores adicionales tales como contenido de energía específica, tasa de emisión de calor, facilidad de almacenamiento y manipulación, limitan significativamente la elección. La liberación de calor de un cohete de combustible líquido puede ser logrado a través de la inyección por separado, la mezcla y combustión de dos líquidos - combustible y oxidante – como en un sistema bi-propelente, o la descomposición exotérmica de tal como hidracina o peróxido de hidrogeno. A diferencia, el cohete de combustible sólido es un diseño relativamente sencillo, el propelente se almacena dentro de la cámara de combustión en forma de granos con formas delimitadas por las paredes de la cámara. Una vez encendido el propelente, la combustión continúa hasta que todo el propelente es consumado. La relación de tiempo - empuje se fija entonces por la configuración geométrica del grano y a su vez, por la configuración que tenga la tobera del motor en el cual se usa el propelente.[11]

[10] Stark, Jhon; Swinerd, Graham y Fortescue Peter. Spacecraft Systems Engineering. Inglaterra: Wiley, 2003. p. 172.
[11] Stark, Jhon; Swinerd, Graham y Fortescue Peter. Spacecraft Systems Engineering. Inglaterra: Wiley, 2003. p. 182.

1.4.1.1. Propelente Líquido

Dos clases importantes de motor se puede encontrar donde el combustible y el oxidante son suministrados por turbo bombas, son diferenciados por su descripción como diseños de ciclo abierto o cerrado. Las configuraciones que se mencionan emplean un elemento de refrigeración regenerativa. El combustible o el oxidante puede ser utilizado como un refrigerante, que fluye a través de una camisa que rodea a la cámara.[12]

1.4.1.2. Propelente Solido

Los propelente sólidos son típicamente de dos tipos: o bien doble base, que comprende mezclas homogéneas coloidales de nitrocelulosa y nitroglicerina, o compuestos que comprenden mezclas de un combustible orgánico y la sal cristalina. El perclorato de amonio, es el principal oxidante utilizado en propulsores compuestos con aglutinante de combustible de polímero, comúnmente poliuretano o polibutadieno. Los cohetes de combustible sólido son relativamente inflexibles en su diseño[13]. Se pueden clasificar de acuerdo a su geometría del grano, ya que una vez se encienda, la variación del consumo del propelente determinara el empuje en un determinado tiempo.

Un grano propelente se dice que es neutral si el empuje se mantiene prácticamente constante a lo largo de la combustión, con una superficie de quemado no varía a través del tiempo; tal comportamiento sería característico de geometría tipo· cigarrillo. En la práctica, una geometría cilíndrica en la que la superficie de quemado sea pequeña, tendrá un empuje limitado.

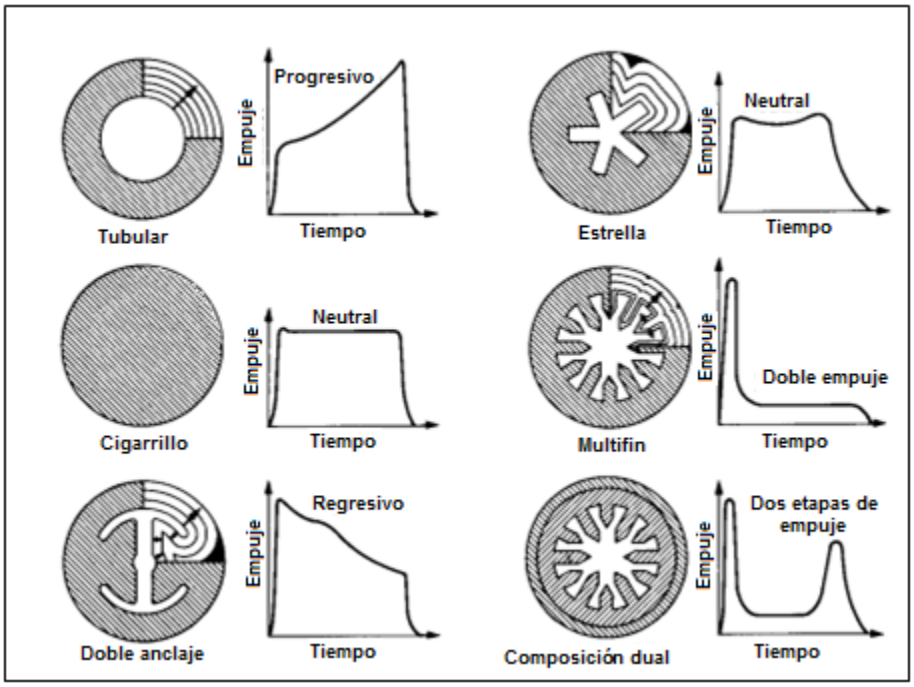
Un grano anular o tubular, por otro lado, es en el que la combustión, es progresiva desde la superficie interior hacia la exterior. En esta configuración, durante la combustión, la superficie de quemado aumenta con el tiempo, y por lo tanto el empuje también lo hace.Una gran superficie inicial de quemado, combinada con un comportamiento de empuje neutral, es proporcionada por el grano cilíndrico en forma de estrella.

El grano que en su interior tenga un cambio de geometría, dará lugar a la combustión de dos etapas, en la que se tendrá la combinación de aspectos tanto de comportamiento progresivo como regresivo, este último acompaña a una disminución de la superficie de quemado.[14]

[12]Ibid p. 186-187
[13]Ibid p. 187
[14]Ibid p. 187

Las características de la relación tiempo-empuje de las configuraciones de grano anteriormente descritas, se muestran en la imagen 1.

Imagen 1.Geometrías de grano de propelente sólido.



Fuente: PETERSON. Mechanics and Thermodynamics of Propulsion. 1992, p 600.

El ARISTARCO será propulsado por medio de un propelente solido Candy Amateur con geometría tipo cigarrillo, se ha determinado este diseño por facilidades de fabricación debido a que la Fundación Universitaria Los Libertadores posee las instalaciones, insumos y el conocimiento para la elaboración de este propelente.

1.5. Estructura

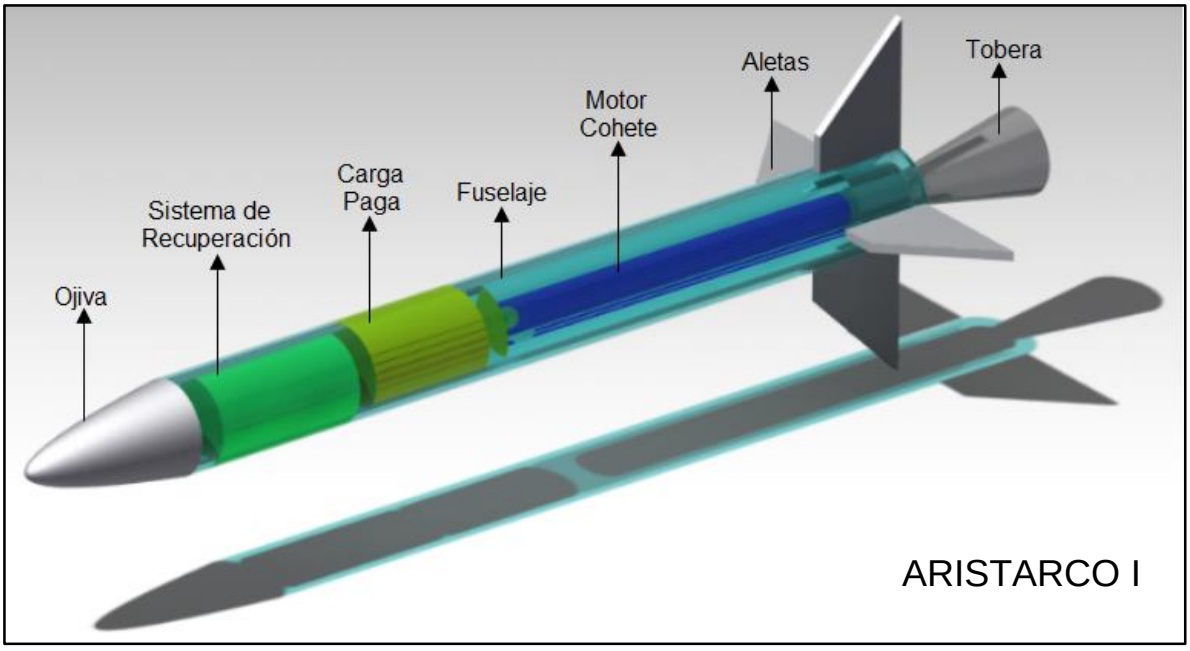
Los requisitos y métodos del diseño estructural de un cohete sonda se basan con mayor criterio en el peso mínimo, vibraciones y consideraciones de material utilizado en vehículos caracterizados por sus misiones espaciales[15]. Los principales objetivos de peso mínimo y máxima fiabilidad se deberán cumplir con un costo mínimo y ajustado al calendario del desarrollo del cohete sonda ARISTARCO I. La estructura es dependiente de otros subsistemas, tales como Análisis Térmico (AT), Determinación de Apogeo (AG), Propulsión (SP), Energía

[15] Stark, Op., Cit., p. 241.

Eléctrica (E), Instrumentación (TR) y finalmente un criterio relevante es la Misión para la cual es construido el cohete sonda. El diseño estructural no solo abarca la selección del material y la configuración, también debe tener un análisis y verificación como parte del proceso.

Para el desarrollo estructural del cohete sonda ARISTARCO I, se han caracterizado subsistemas mostrados en la imagen 2.

Imagen 2. Estructura del Cohete Sonda ARISTARCO I.



Fuente: Autores

1.5.1. Ojiva

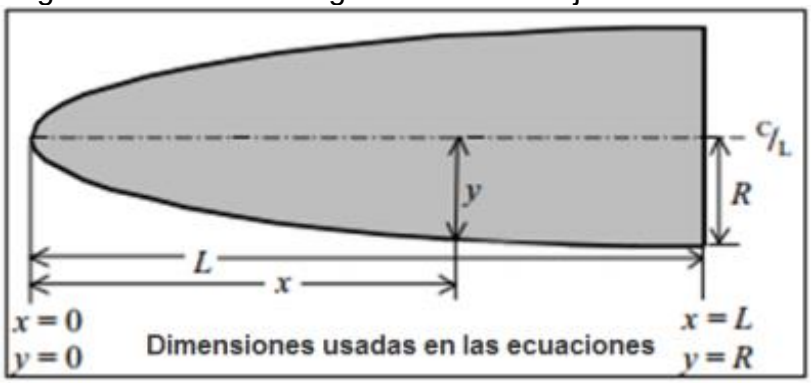
La ojiva es definida como “una figura formada por dos arcos que se cortan en ángulo”, su objetivo es sellar la parte frontal del cuerpo del cohete y así disminuir la resistencia que el aire opone al movimiento del cohete[16]. La magnitud de la resistencia que el aire opone al movimiento del cohete (*Drag*), depende fundamentalmente de la geometría de la ojiva, el diámetro del fuselaje del cohete y la velocidad del mismo. El primer punto de contacto que tienen las partículas de flujo del aire con el cohete es el extremo frontal de este, lugar que ocupa la ojiva. Por esta razón la geometría que posee la ojiva es la principal característica a tener en cuenta para la elección del tipo de ojiva de un cohete en general. Ya que la

[16]Velez, Hector. ESTUDIO AERODINÁMICO DE LA OJIVA DEL COHETE SONDA LIBERTADOR I MEDIANTE SOFTWARE CFD. Bogota D.C, 2013. p. 22.

fuerza de *Drag* que se presentara sobre este depende de la velocidad que lleve el cuerpo, del coeficiente de *Drag* del mismo y de la superficie que tiene contacto con el aire, y por ende la geometría y dimensiones del cuerpo.[17]

Existen diversos tipos de ojivas para cohetes, cuya clasificación está basada en la configuración geométrica que presentan. Todas las ojivas, poseen dimensiones generales (Ver imagen 3.) que las caracterizan y conforman su configuración geométrica. Por facilidades de interpretación de imágenes y ecuaciones posteriores, L será la longitud total de la ojiva, R el radio de la base de la misma, Y el radio en un punto cualquiera determinado por X, que varía desde el extremo frontal de la ojiva (punto 0) sobre la longitud L.[18]

Imagen 3. Dimensiones generales de la ojiva de un cohete.



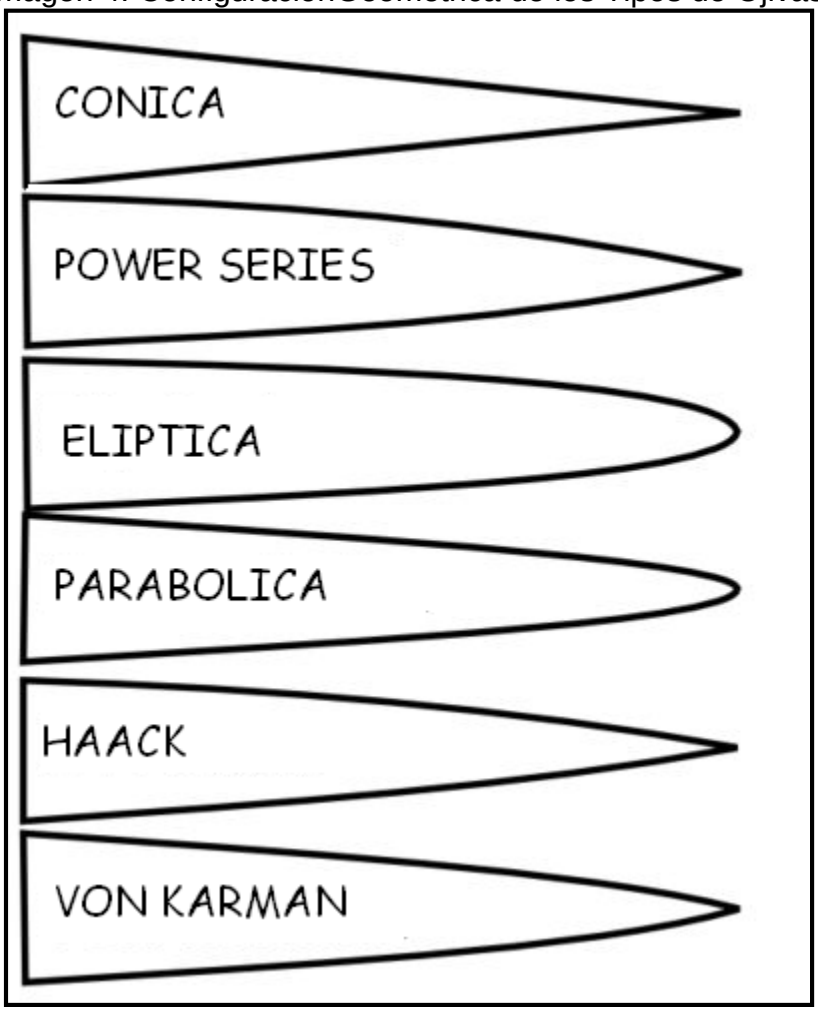
Fuente: Gary A. Crowell Sr. The Descriptive Geometry of Nose Cones. [en línea]

1.5.1.1. Tipos de Ojivas

En la imagen 4, se muestran las configuraciones geométricas de los tipos de ojivas. En esta se encuentran ojiva cónica, power series, elíptica, parabólica, Haack y Von Karman, respectivamente.

[17] Science Learning, Rocket Aerodynamics; [en línea].
[18] Velez, Op., Cit., p. 29.

Imagen 4. ConfiguraciónGeométrica de los Tipos de Ojivas.



Fuente: Design a Rocket Nose Cone. [En línea].

1.5.1.1.1. Ojiva Cónica

La ojiva cónica es usada comúnmente, debido a la simplicidad de su forma geométrica, ya que esta es un cono; adicionalmente es frecuentemente usada debido a la facilidad de fabricación.[19]

1.5.1.1.2. Ojiva Power Series

La ojiva power series, geométricamente está compuesta por un cono parabólico, formado por la rotación de la parábola sobre su propio eje; se caracteriza por

[19]Gary. THE DESCRIPTIVE GEOMETRY OF NOSE CONES.1996.

poseer una punta redonda y porque su base no es tangente al cuerpo cilíndrico del vehículo, generando una discontinuidad entre la unión de la ojiva y el fuselaje del cohete, lo que produce la sensación de no ser adecuada aerodinámicamente.[20]

1.5.1.1.3. Ojiva Elíptica

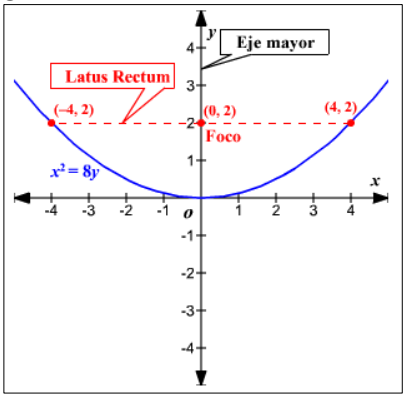
El perfil de la ojiva elíptica, está definido por la mitad de una elipse. La revolución total de una elipse sobre su eje principal, es denominada prolato de esferoide, por lo que el nombre de esta ojiva debería ser prolato de semiesferoide. Generalmente, es usada en modelos de cohetería que operan a velocidades de régimen subsónico. Profesionalmente en cohetería, estas ojivas no son normalmente usadas, debido a que tienen mejor desempeño en régimen subsónico.[21]

1.5.1.1.4. Ojiva Serie Parabólica

La forma de la nariz de la serie parabólica no es la forma contundente que se visualiza cuando comúnmente se refieren a un cono de nariz 'parabólica'. La forma de la nariz Series parabólica es generada por la rotación de un segmento de una parábola alrededor de una línea paralela a su Latus Rectum.[22]

La latus rectum de una sección cónica es la cuerda (segmento de línea) que pasa a través del foco, es perpendicular al eje mayor y tienen ambos puntos finales en la curva,[23] (ver Imagen 5).

Imagen 5. Definición Latus Rectum.



Fuente: Hotmath.com, LATUS RECTUM [en línea]

[20] Velez, Op., Cit., p. 32.
[21] Velez, Op., Cit., p. 31.
[22] Gary, Op., Cit., p. 5.
[23] Hotmath.com, Latus Rectum; [en línea].

La construcción de este tipo de ojiva, produce una forma de la nariz con una punta afilada. Para la forma contundente típicamente asociados con una ojiva 'parabólica', ver la ojiva Power Series.[24]

1.5.1.1.5. Ojiva Serie Haack y Von Karman

En el tipo de ojivas serie Haack y Von Karman, a diferencia de todas las formas de ojivas anteriores, no se construyen a partir de figuras geométricas. Las formas en su lugar son derivadas matemáticamente con el fin de minimizar el arrastre. La ojiva serie Haack y Von Karman es un conjunto continuo de formas determinadas por el valor de C, donde tienen un significado especial; cuando C=0, la notación "LD" indica, mínima resistencia para la longitud y el diámetro dado, y cuando C=1/3, "LV" indica mínima resistencia para una longitud dada y volumen dado. El termino C, es una constante de las ojivas serie Haack y Von Karman, para las cuales tiene un valor de 0 y 1/3, respectivamente. [25]

En un estudio aerodinámico realizado a múltiples tipos de ojivas para cohetes, se obtiene como resultado el coeficiente de *Drag*de las múltiples ojivas, operando a diferentes regímenes de velocidades, (subsónico, transónico y supersónico) entre Mach= 0.01 y Mach=3.8. El análisis aerodinámico mencionado, fue realizado mediante la simulación en software CFD. A partir de las ecuaciones pertinentes, se determinaron las dimensiones de cada ojiva a evaluar, posteriormente fueron modeladas en CAD y por último, evaluadas en la simulación. A partir de una propiedad que posee el software utilizado se determinó el área expuesta al flujo de cada ojiva bajo análisis. A través de este valor, de los resultados de la fuerza aerodinámica (*Drag*) que actúan sobre el objeto estudiado, y de los parámetros de evaluación del software al nivel medio del mar como densidad del aire, los valores de velocidad utilizados, despejando de la ecuación (2) *Drag*, se calculó el coeficiente de *Drag* (Cd), y posteriormente, mediante los valores obtenido de Cd, se obtiene la gráfica de Cd vs Mach.[26]

Ecuación 2.Ecuación Fuerza de *Drag*

$$D = \frac{1}{2} * \rho * v^2 * Cd * S_u$$

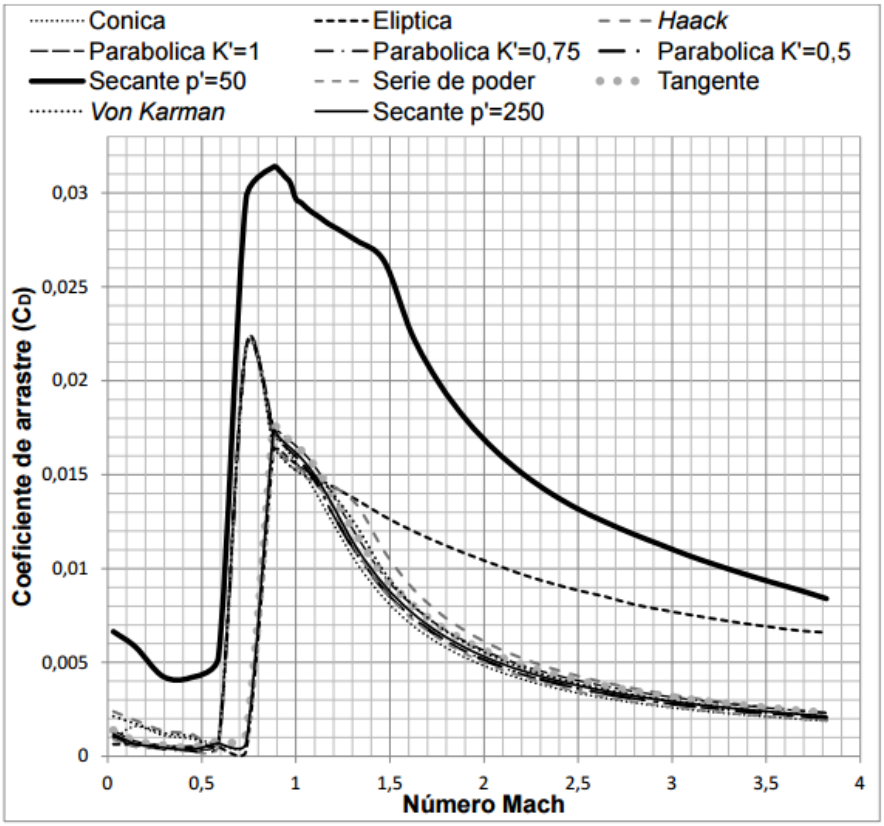
Fuente: NASA, The *Drag* Equation; [en línea].

[24] Gary, Op., Cit., p. 5.
[25] Velez, Op., Cit., p. 35.
[26] Velez, Op., Cit., p. 97.

Donde (D)es la fuerza de resistencia del aire (Drag), (ρ) es la densidad del fluido, (v) es la velocidad del cuerpo inmerso en el fluido, (Cd) es el coeficiente de drag del cuerpo y (S_u) es la superficie del cuerpo.

En la gráfica 1, están evidenciados los resultados obtenidos del Cd contra el número de mach en el cual fueron obtenidos. En dicha grafica se puede observar que el mayor valor para coeficiente de *Drag*, está en la región transónica (0,8-1 Mach), siendo tendencia para todos los tipos de ojiva evaluados. De acuerdo al desempeño observado en la gráfica a través de las diferentes ojivas evaluadas, se evidencia que la mayoría de ojivas descritas anteriormente, tienen un comportamiento muy similar en el que el valor máximo de Cd es de 0,0225. Adicionando el factor de facilidad de fabricación, se determina implementar en el ARISTARCO I la ojiva tipo Elíptica, la cual, posee un valor máximo de Cd de aproximadamente 0,175, lo que significa que es un valor aceptable para el desempeño aerodinámico.

Grafica 1.Coeficiente de arrastre (Cd) Vs Número de Mach.



Fuente: Gary A. Crowell Sr. The Descriptive Geometry of Nose Cones. [en línea]

1.5.2. Fuselaje

Es la estructura aerodinámica del vehículo encargada de contener todos los subsistemas que conforman el cohete, como son: el sistema de recuperación, carga paga, sistema de propulsión. Exteriormente se encuentran acoplados al fuselaje el subsistema estabilizador y la ojiva, encargados de proporcionar el desempeño y las características aerodinámicas principales.[27]

El fuselaje es una parte esencial del cohete, ya que este abarca la mayor parte del cuerpo del vehículo; y por ende el desempeño del cohete depende directamente de este. Su función dinámica, es transmitir la carga de las fuerzas generadas durante el vuelo; en el que se busca generar el menor arrastre aerodinámico posible.[28]

Estructuralmente, la configuración cilíndrica del fuselaje es altamente eficiente, ya que dicha forma distribuye uniformemente las cargas presentadas sobre el cuerpo, disminuyendo así la posibilidad de generación de zonas de concentración de fallas.

Generalmente los materiales implementados para la construcción de este tipo de cohetes, son la fibra fenólica, PVC y aluminio, debido a que estos materiales presentan bajo peso y aceptables condiciones de resistencia. Para el ARISTARCO I, se decide utilizar fibra fenólicapara el cuerpo del fuselaje, debido a que comercialmente existe mayor gama de diámetros de los cilindros que los ofrecidos en PVC y aluminio.[29]

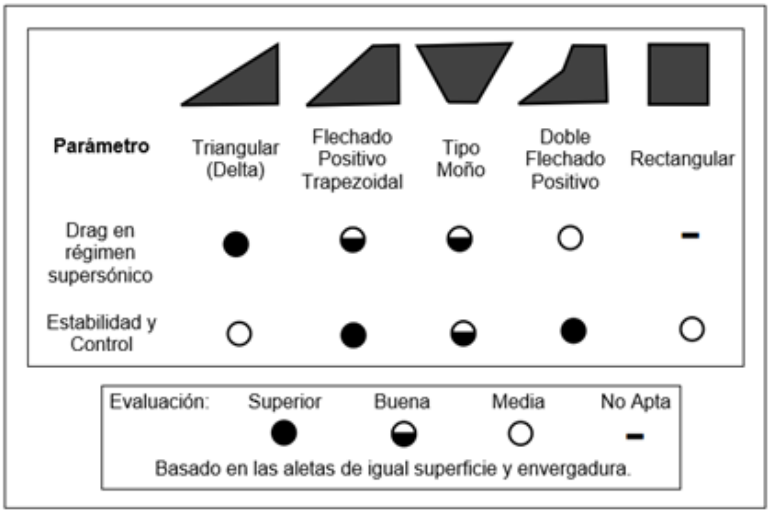
1.5.3. AletasEstabilizadoras

El tipo y tamaño del dispositivo estabilizador puede variar de acuerdo a las necesidades requeridas por el cohete sonda; en la imagen 6 se puede evidenciar las diferentes alternativas de geometrías utilizadas para estabilizar un cohete. La imagen 6, es una comparación entre la aleta triangular (delta), trapezoidal convencional con “flechado positivo”, Tipo “Moño”, doble “flechado positivo”, y una superficie con geometría rectangular[30]. La comparación es basada en dos

[27] Riveros Felipe, y Rodríguez Alejandro. Diseño y construcción de un cohete aficionado controlado mediante el accionamiento de una tobera de empuje vectorial. Universidad Militar Nueva Granada. Facultad de Ingeniería, 2010, p. 7.
[28] Fuselaje, Tecnología de Materiales; [en linea].
[29]Ibíd p. 7.
[30] Eugene L. Flemman. Tactical Missile Design, Second Edition. Virginia, 2006. p. 51.

parámetros, el primero es el *Drag* en régimen supersónico, y el segundo, es estabilidad y control.

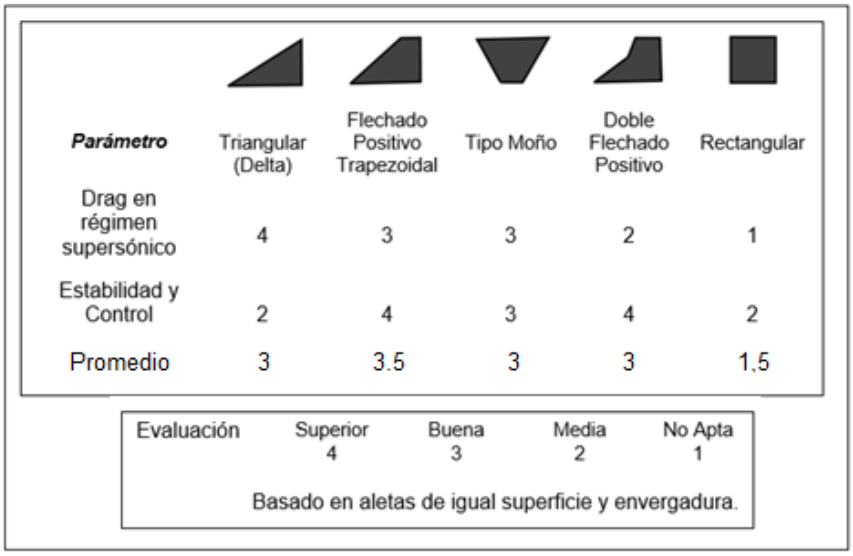
Imagen 6. Comparación cualitativa de alternativas para la geometría de aletas



Fuente: Fleeman. Tactical Missile Design. 2006, p.52.

En base a la comparación cualitativa (imagen 6) se decide asignar un valor numérico a cada una de las variables evaluativas con la finalidad de obtener un valor promedio para cada uno de los parámetros y así generar una mayor confiabilidad en la selección de aletas del cohete sonda ARISTARCO I, dichos valores, se encuentran registrados en la imagen 7.

Imagen 7. Comparación cuantitativade alternativas para la geometría de aletas

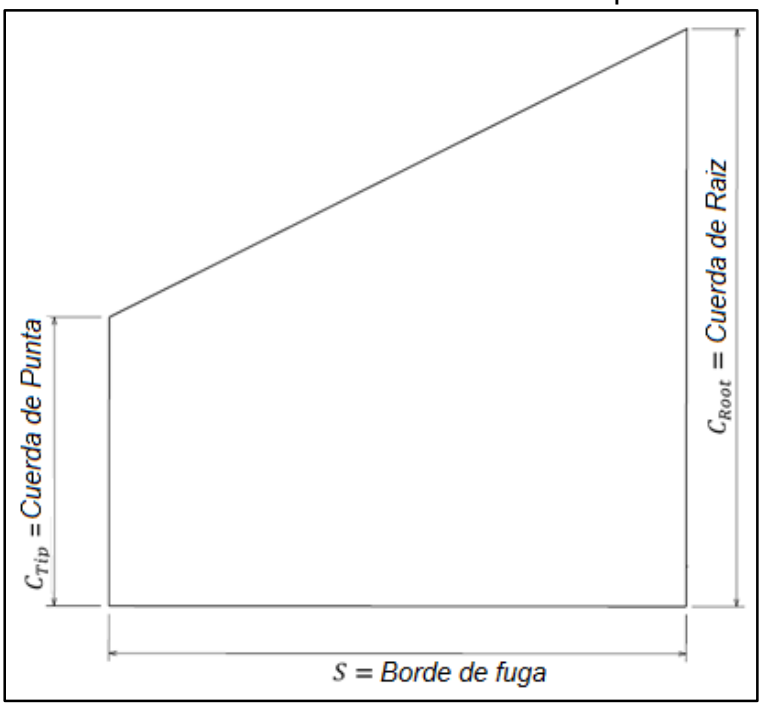


Fuente: Fleeman Tactical Missile Design. 2006.

Luego de observar los cuadros comparativos cualitativos y cuantitativos (Imagen 6 y 7 respectivamente), se ha escogido para el cohete sonda ARISTARCO I el estabilizador con geometría flechado positivo trapezoidal de acuerdo a que presenta las mejores condiciones en las cuales fue evaluado con respecto a los demás estabilizadores, siendo la geometría con mejor promedio de evaluación.

El estabilizador con flechado positivo trapezoidal contiene una designación de variables para su diseño preliminar y detallado; se pueden observar en la siguiente imagen, las variables a tener en cuenta para el posterior desarrollo de diseño preliminar en la sección 2.2.3.

Imagen 8. Características de estabilizador flechado positivo trapezoidal

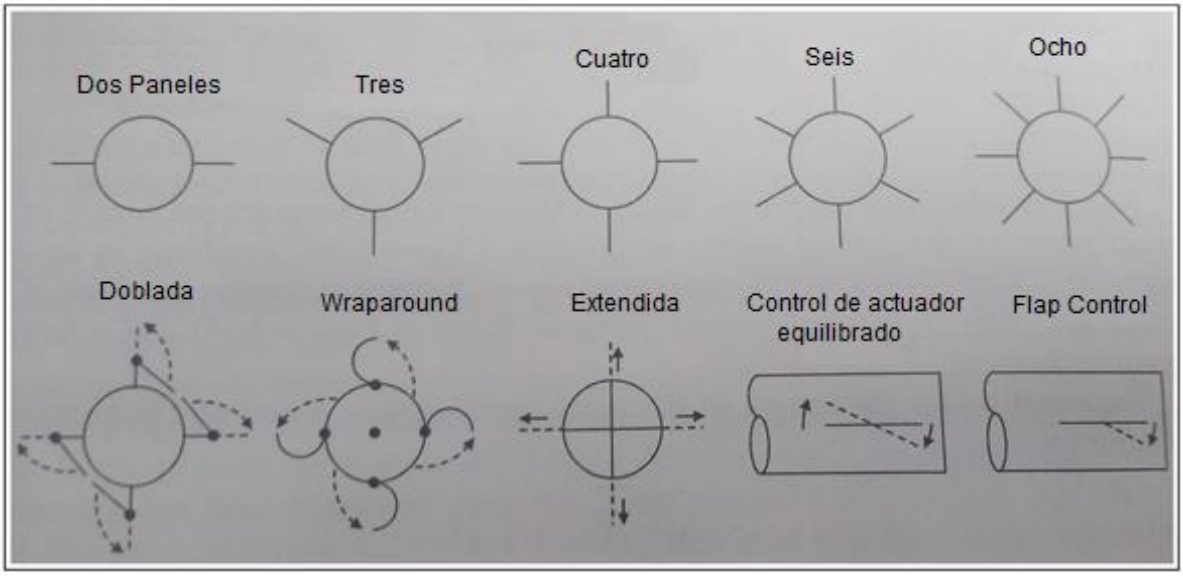


Fuente: Autores

La geometría del estabilizador de un cohete usualmente tiene gran relevancia en términos como estabilidad aerodinámica, control, y rendimiento de este, otro factor influyente en el sistema de estabilizador es la cantidad de aletas que puede contener el cohete, y las alternativas de integración de la superficie de control. El número de superficies puede variar desde dos aletas hasta ocho aletas, los esquemas de plegado se contemplan tres (*Folded, Wraparound, Extended*), y finalmente la deflexión de la superficie de control en la cual se han considerado

dos opciones (*Balanced Actuaction Control, Flap Control*)[31], como se puede observar en la imagen 9.

Imagen 9. Configuración de Aletas.



Fuente: Fleeman. Tactical Missile Design. 2006, p.52.

1.5.4. Carga Paga

El sistema de carga paga o también llamado sistema de carga útil, depende directamente de la misión del cohete en el cual va implementado el sistema. A través de la historia la carga útil de los cohetes, ha ido variando según los requerimientos que se tengan de acuerdo al marco de referencia de la época; inicialmente la carga paga de un cohete, era una carga de fuegos artificiales para la celebración de festividades, posteriormente, durante la segunda guerra mundial (WWII), Alemania fue pionera en implementar explosivos en la carga paga contenida en estos vehículos; continuando con la modificación de los cohetes para realizar el lanzamiento y puesta en órbita de satélites para cumplir misiones como de comunicaciones, monitoreo del tiempo, espionaje, exploración planetaria y observación. Adicional a las ya mencionadas, la carga paga de cohetes, también ha estado constituida por seres vivos, plantas, animales y seres humanos, siendo estos últimos la carga paga más importante transportada en un vehículo aeroespacial.[32]

El ARISTARCO I, tiene como misión transportar su carga paga (PL) a una altitud mínima de un (1) Kilometro. Esta PL, tendrá como fin, el realizar la toma de datos de las características atmosféricas a la altitud de apogeo, la cual es determinada

[31] Fleeman, Op., Cit., p. 54.
[32]NASA, Palead Systems; [En línea]

en la etapa “2. DISEÑO PRELIMINAR”. Por lo tanto, la PL del ARISTARCO I, estará compuesta por un dispositivo de medición y adquisición de los datos atmosféricos ya mencionados, descrito detalladamente en la sección Instrumentos (TR). Sin embargo, en el cohete ARISTARCO I, podrán ser implementados otros sistemas de PL, debido a la versatilidad del vehículo, podrá ser instalada una PL con otros fines, siempre y cuando cumpla con las características de dimensionamiento y peso adecuadas para el cohete en cuestión, las cuales son establecidas en la sección “2. DISEÑO PRELIMINAR”.

1.5.5. Motor

El motor de un cohete se condiciona por varios procesos químicos y físicos durante su operación los cuales son altamente complejos. Estos procesos incluyen las complejas reacciones químicas que ocurren durante la combustión; la forma en que se consume el propelente durante este proceso; el comportamiento del flujo de los gases de escape a medida que se forma en la superficie de combustión, viaja a través de la cámara y sale a través de la tobera; la interacción entre los gases de escape y las partículas condensadas (humo).[33]

El motor de un cohete se integra por 3 partes las cuales corresponden a:

- El propelente (P): El propelente utilizado en los motores cohete amateur experimentales puede ser una composición simple, siendo la combinación de dos constituyentes principales – combustible y oxidante. Tal es el caso de los propelentes con bases de "azúcar".[34]

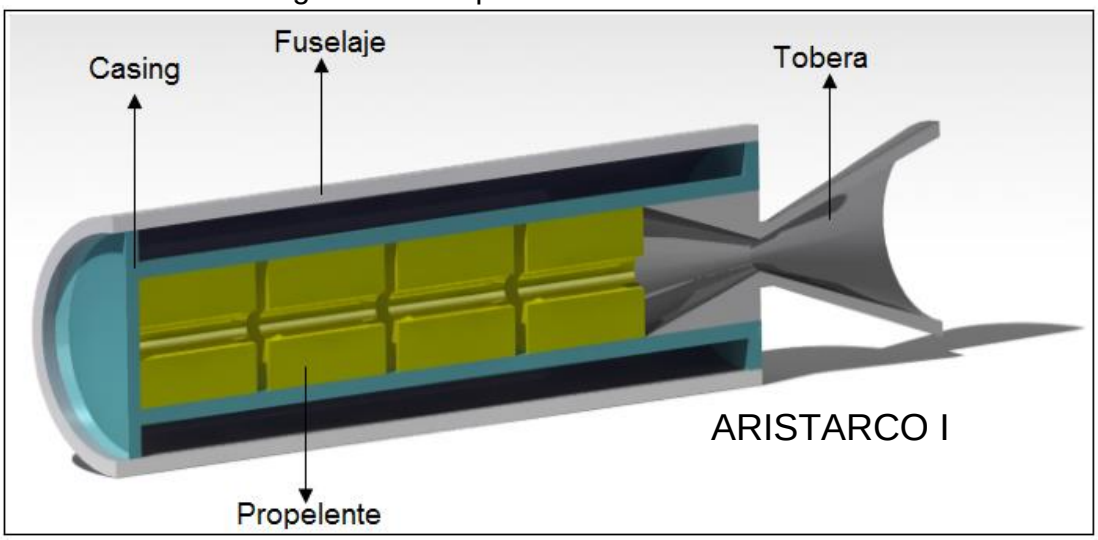
- La tobera (To): La función primaria de una tobera es canalizar y acelerar los productos de la combustión producidos por el propelente de tal manera que maximice la velocidad del escape de salida, a una velocidad supersónica. La tobera familiar de un cohete, conocida como convergente-divergente o tobera de Laval, cumple con esta característica con una simple geometría.[35]

- La cámara de combustión (Cas): Es la sección donde se introduce el propelente sólido y se ajusta la tobera en la salida de dicho componente, el cual tiene una geometría similar a la de un cilindro como se puede observar en la siguiente figura; otra función que cumple la cámara de combustión en el cohete ARISTARCO I, es permitir el ensamble con el fuselaje, con un

[33] Richard Nakka. Teoría sobre motores cohete de propelente sólido. [en línea]
[34] Ibíd.
[35] Ibíd.

soporte superior e inferior, mediante los cuales se genera un espaciamiento el cual se encuentra citado en la sección de Aislante Térmico (AT).[36]

Imagen 10. Componentes del motor cohete.



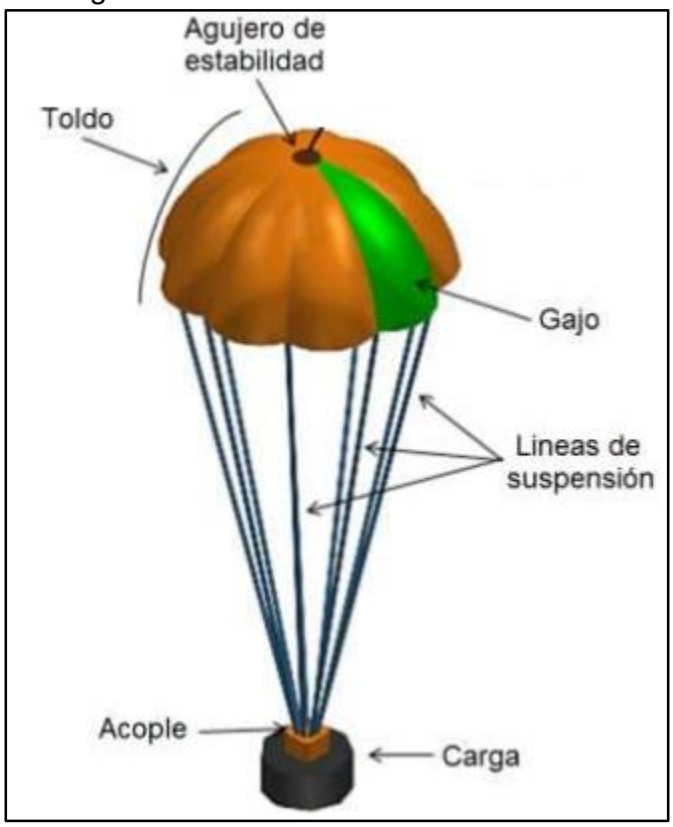
Fuente: Autores

1.6. Sistema de Recuperación

Un sistema de recuperación se compone de un conjunto de elementos, los cuales funcionando juntos, prevé la desaceleración controlada y la estabilización de un cuerpo en vuelo. Las características principales de un dispositivo de desaceleración son el*Drag* y estabilidad[37]. Los elementos característicos de un sistema de recuperación se pueden visualizar en la siguiente imagen.

[36] Nakka. Op., Cit.
[37] E. G. Ewing, H.W. Bixby y T.W. Knache. Recovery system design guide. Gardena, California, 1978. p. 73.

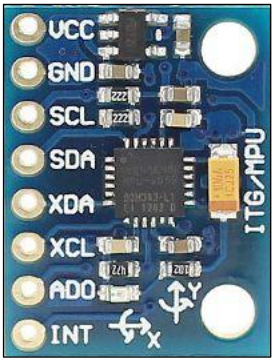
Imagen 11. Configuración característica del sistema de recuperación.



Fuente: Castillo, Leudy. Jiménez Nixon. DISEÑO, DESARROLLO Y PRUEBAS DEL SISTEMA DE RECUPERACION PARA EL COHETE SONDA LIBERTADOR I.Bogotá D.C, 2014. p. 57.

Una de las características relevantes de la computadora de vuelo del cohete sonda ARISTARCO I, es que permite accionar el sistema de recuperación, garantizando el despliegue de este en la sección que se encuentra entre la unión de la ojiva y el fuselaje, mediante el sensor MPU-6050, mostrado en la imagen 12.

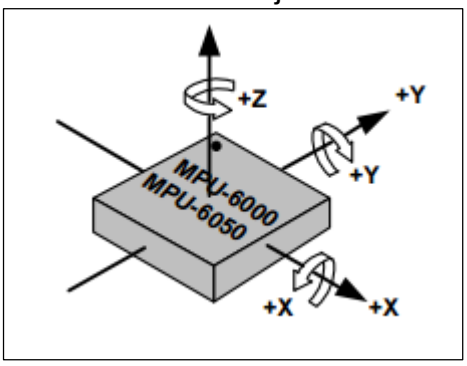
Imagen 12. Sensor MPU-6050.



Fuente: Electrónica Practica Aplicada,SENSOR MPU-6050 [en línea].

Este sensor es un acelerómetro y giroscopio, el cual realiza mediciones en cada uno de los 3 ejes en los cuales tiene libertad de movimiento el cohete (*X, Y* y *Z*), como se observa en la imagen 13.[38]

Imagen 13. Orientación de los ejes del sensor MPU-6050.



Fuente: InvenSense. MPU-6000 and MPU-6050 Product Specification [en linea].

El sistema de recuperación del ARISTARCO I, es accionado cuando el sensor detecta que la inclinación del cohete ha sobrepasado un ángulo en específico con respecto al eje Y o X (Ver sección 2.7.). El accionamiento de dicho sistema, consiste en activar dos motores, donde cada uno moverá una cremallera que se encarga de eyectar la ojiva y el paracaídas del vehículo.

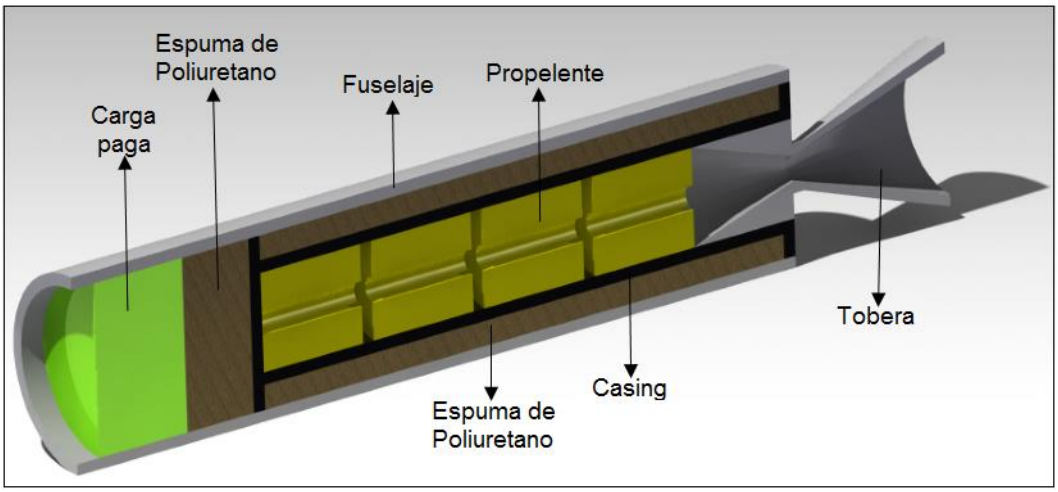
1.7. Aislante Térmico

Para realizar el aislante térmico en el interior del sistema estructural del cohete sonda ARISTARCO I, se empleará un método de espaciamiento entre la fuente de aumento de temperatura (P)y el fuselaje del cohete, así como entre la sección superior del sistema de propulsión y la carga paga (PL), con fin de evitar la conducción térmica entre los componentes ya mencionados. Para disminuir los efectos causados por la trasferencia de calor mediante radiación, los espaciamientos mencionados son ocupados por un material que presenta características de aislamiento térmico como se muestra en la imagen 15; para cumplir lo anterior se empleara Espuma de Poliuretano (PUR), el cual es un material sintético obtenido a partir de la mezcla de dos componentes generados mediante procesos químicos derivados del petróleo y el azúcar (Isocianato y Poliol)[39], este fue seleccionado ya que posee una elevada capacidad de aislante

[38] InvenSense. MPU-6000 and MPU-6050 Product Specification. [En línea].
[39] El aislamiento y el PUR. Características y aplicaciones [en línea]

debido a la baja conductividad térmica que posee el gas espumante la cual puede situarse alrededor de 0,022 W/(m*K), dicho valor aumenta con el paso del tiempo hasta 9 meses después de envejecimiento, estabilizando el valor en 0,028W/(m*K); lo anterior supone un 25% de mayor eficiencia con respecto a la media de otros productos utilizados de aislamiento térmico como lana mineral o espuma de poliestireno extruido y expandido[40]. Adicional al desempeño como aislante térmico se determina la implementación de este material ya que comercialmente es fácilmente accesible, económicamente viable, y una alta maleabilidad, lo que genera facilidad en el moldeo y por ende una mejor adaptación en estos espacios.

Imagen 14. Ubicación del aislante en el ARISTARCO I.



Fuente: Autores

1.8. Instrumentos

La instrumentación del ARISTARCO I, se conforma por dispositivos de medición, los cuales corresponden a: un temporizador, un sensor de presión atmosférica, sensor de temperatura exterior, mediante los cuales, se puede realizar la determinación del apogeo y la altitud del vehículo en un tiempo determinado, a través de la computación de los datos obtenidos por cada uno de los sensores mencionados, los cuales se encuentran implementados en el sensor BMP-180, el cual es mostrado en la imagen 15.

[40] Ibíd

Imagen 15. Sensor BMP 180.

Fuente: Electronilab. Sensor BMP-180 [en línea].

1.9. Operación y sistema en tierra

La función del sistema de operación y equipo en tierra, consiste en la combinación de las acciones e implementos que permitan llevar a cabo la fase inicial (lanzamiento), y la fase final (aterrizaje y recuperación) de la misión del cohete sonda ARISTARCO I; principalmente este sistema está comprendido por el subsistema ignitor y el localizador del vehículo.

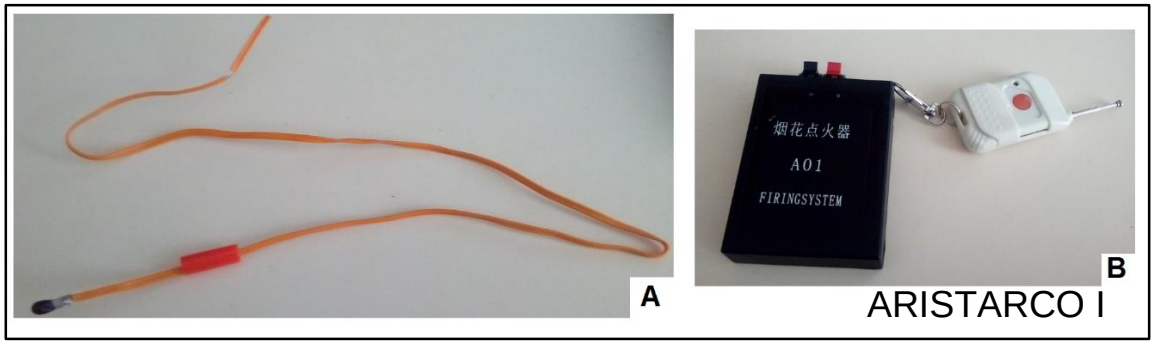
El localizador que se implementara en el cohete *ARISTARCO I,* es el localizador *Trackerpad,* el cual es un localizador satelital, del tamaño de una moneda y batería de 7 días de autonomía, que será instalado dentro del sistema de recuperación del cohete, cuyo sistema permiteubicar el vehículo a través de un teléfono celular (*Android*).[41]

El lanzamiento del vehículo, se lleva a cabo mediante la ignición del motor cohete, el cual al ser un motor cohete de propulsor sólido, se inicia a través de un dispositivo pirotécnico que consiste en un estopín o también llamado fosforo eléctrico (Un dispositivo eléctrico que contiene una pequeña cantidad de material pirotécnico que se enciende cuando la corriente fluye a través del dispositivo[42]), el cual tiene contacto con un alambre de resistencia eléctrica y cuya ignición es realizada a distancia a través de un detonador (ver imagen 16 A) conectado a dicho alambre, cuyo funcionamiento es iniciado por un control remoto (ver imagen 16 B), cuyo alcance es de 10 metros de distancia. El inicio de la quema del estopín, desarrolla altas temperaturas que diseminan el grano del propelente sólido y provoca la explosión del combustible.[43] Para este fin, se ha seleccionado

[41] DIGITAL TRENDS. Trackerpad. [En línea].
[42] Glossary of Pyrotechnic Terms. [En línea]
[43] Gomez, Fabian. Leiva, Huindi. ANÁLISIS DEL RENDIMIENTO DEL PROPELENTE SOLIDO TIPO AMATEUR MEDIANTE BALLISTIC EVALUATION MOTOR (BEM) Y SELECCIÓN DE LA

un subsistema ignitor disponible en el mercado, denominado *A01 FIRINGSYSTEM*.

Imagen 16. (A) Estopín, (B) Detonador y Control.



Fuente: Autores

1.10. Energía Eléctrica

La energía eléctrica requerida para la operación del ARISTARCO I, se centra únicamente en dos (2) sistemas: El sistema de carga paga (PL) para el funcionamiento de la instrumentación comprendida en la computadora de vuelo en la sección de instrumentos (TR), y el sistema de operación y sistema en tierra (OT), para la fase de ignición de lanzamiento del vehículo.

Para desarrollar la computadora de vuelo, se utilizará la plataforma ARDUINO, implementando la tarjeta ARDUINO UNO, la cual puede ser energizada por dos vías, la primera es conexión USB, que proporciona 5 voltios y la segunda o por un *jack* de alimentación, cuya fuente de alimentación se recomienda que se encuentre entre 7 y 12 voltios.[44]

Para la fase de inicio de operación (ignición de lanzamiento), según lo estipulado por el fabricante, es necesario implementar cuatro baterías tipo AA.

1.11. Masa de lanzamiento

El objetivo del cohete sonda ARISTARCO I, es llevar a cabo la misión de transporte de un dispositivo de medición de condiciones atmosféricas a una altitud por encima de un kilómetro, la cual es determinada con exactitud en la segunda fase de diseño (DISEÑO PRELIMINAR).

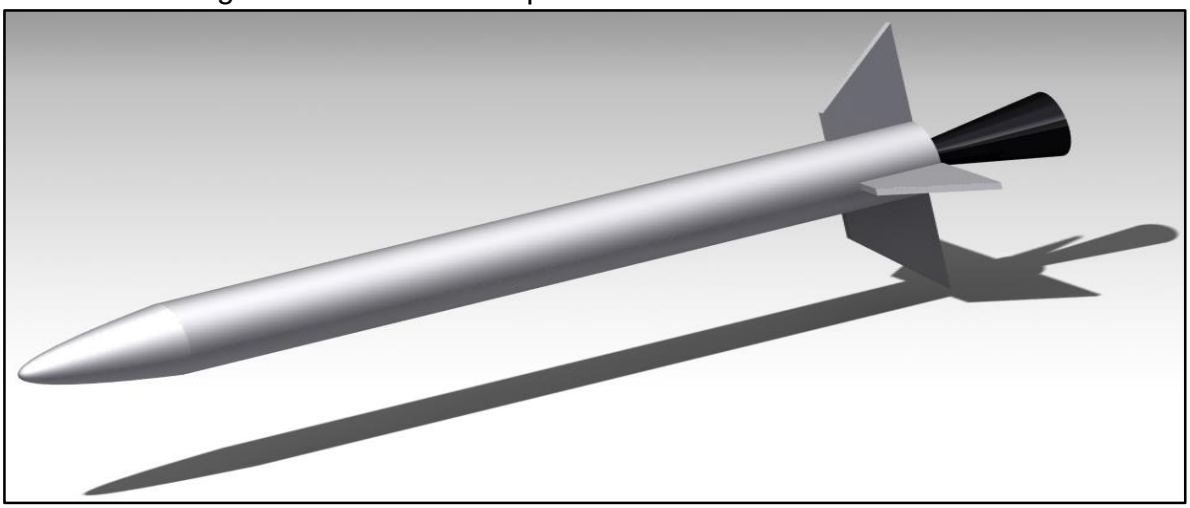
TOBERA MAS ADECUADA PARA SU USO EN EL COHETE SONDA LIBERTADOR I. Bogota DC. 2015. p. 30.

[44] Gallardo Daniel. Apuntes de ARDUINO. [En línea].

El cohete está compuesto por cinco sistemas principales: Sistema de Propulsión, Estructura, Sistema de Recuperación, Sistema de Medición y Adquisición de Datos (Instrumentos TR) y Sistema de Operación en Tierra; cada uno de ellos, con subsistemas que los conforman y complementan su funcionamiento.

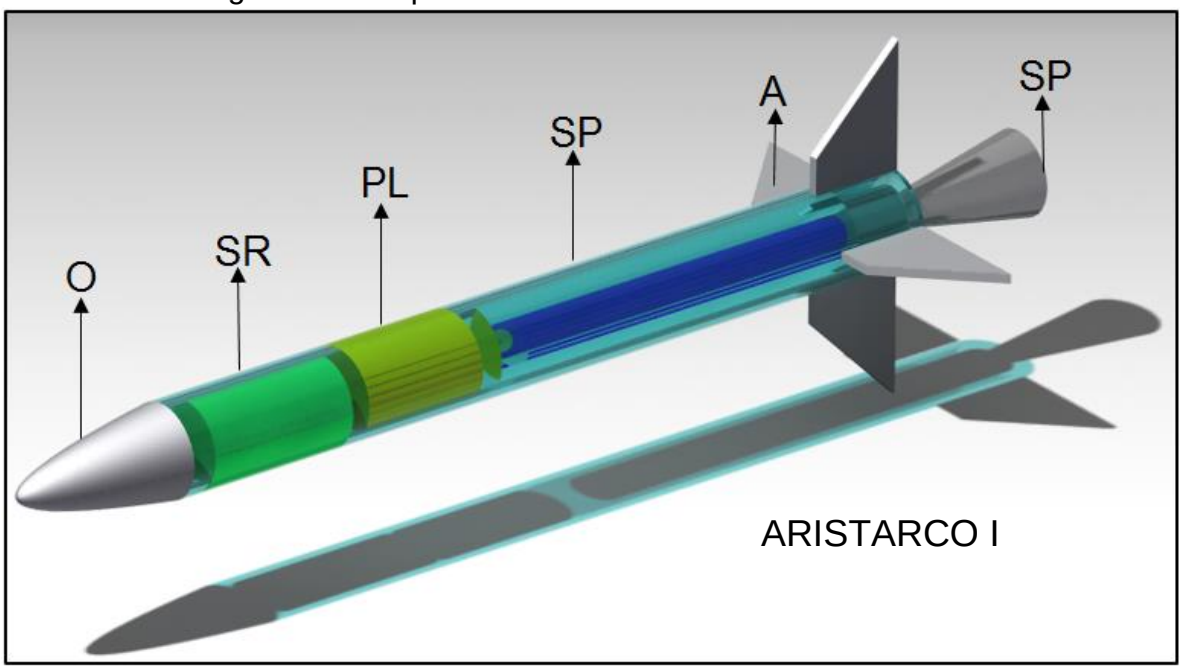
El ARISTARCO I, es un cohete sonda, en el cual, el sistema de propulsión, consiste en un motor cohete de propelente sólido, tipo Candy amateur. La estructura del ARISTARCO I, se compone de una ojiva tipo elíptica, un cuerpo fuselado cilíndrico, cuatro aletas estabilizadoras tipo flechado positivo. El sistema de recuperación es el sistema, mediante el cual es expulsado y accionado el paracaídas seleccionado para la recuperación del vehículo después de finalizada la misión, dichas acciones, son realizadas mediante la computadora de vuelo que posee el cohete. El sistema de medición u adquisición de datos (TR), está conformado por la computadora de vuelo, la cual determina mediante sensores la presión atmosférica, la temperatura, la altitud y aceleración en un periodo de tiempo determinado. El OT se conforma por el subsistema de control remoto-detonador y por el estopín de ignición.

Imagen 17. Diseño conceptual cohete sonda ARISTARCO I.



Fuente: Autores

Imagen 18. Componentes del cohete sonda ARISTARCO I



Fuente: Autores.
(*Ver lista de abreviaturas)

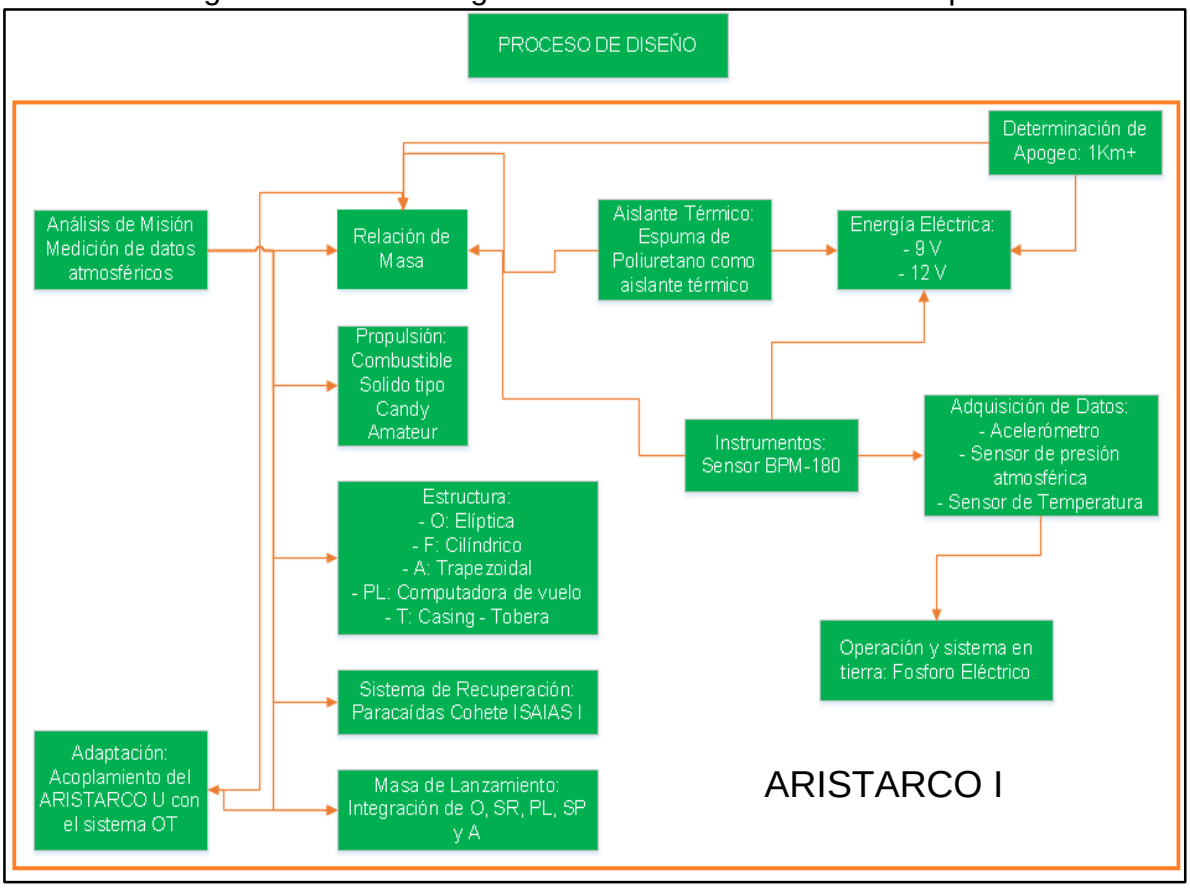
1.12. Adaptación

La adaptación del ARISTARCO I, consiste básicamente en realizar el ensamble de los cinco sistemas que conforman el cohete como tal, Sistema de Propulsión (SP), Sistema de Recuperación (SR), Instrumentos(TR),Estructura (ES) y Operación en Tierra, obteniendo como resultante el cohete ARISTARCO I, en condiciones para llevar a cabo la misión para la cual fue diseñado.

Una vez se realizan los mencionados ensambles y acoplamientos de los sistemas, resulta la combinación de componentes, la cual conforma al ARISTARCO I y permite la ejecución de la misión del mismo.

Según la metodología del proyecto, el diseño conceptual del cohete sonda ARISTARCO I, una vez seleccionados cada uno de los componentes descritos en esta, la fase de diseño 1. DISEÑO CONCEPTUAL, queda evidenciada en el diagrama 3.

Diagrama 3. Metodología desarrollada del diseño conceptual.



Fuente: Autores

2. DISEÑO PRELIMINAR

De acuerdo a las características obtenidas en la fase de Diseño Conceptual se obtiene la metodología desarrollada de dicha fase (Diagrama 3) y teniendo en cuenta los parámetros seleccionados para el cohete sonda *ARISTARCO I*, de acuerdo a estos, en la fase de Diseño Preliminar se evaluarán, a través de cálculos cuantitativos, los ítems caracterizados en la metodología de diseño del cohete sonda *ARISTARCO I* (Diagrama 1).

2.1. Misión

En concordancia con lo estipulado en la fase de Diseño Conceptual la misión del *ARISTARCO I,* consiste en realizar medición y toma de datos de temperatura y presión atmosféricas, a una altitud de operación superior a 1 kilómetro, desde la zona de lanzamiento.

2.2. Estructura

Debido a que es necesario tener el espacio suficiente en el interior del fuselaje del vehículo para almacenar la carga, la cual es diseñada, con base en la tarjeta ARDUINO UNO (Ver sección 2.7.),cuyas dimensiones son 6,86cm de altura y 5,34cm de ancho, se decide utilizar cilindros de fibra fenólica, cuyas dimensiones son Diámetro exterior de 7,2cmy Diámetro interno de 6,935cm, debido a que dichas dimensiones permiten almacenar en su interior la tarjeta anteriormente mencionada.

Para determinar la longitud del fuselaje, es considerado el parámetro de diseño que establece que la relación de longitud-diámetro, debe encontrarse entre 5 y 25.[45]Dado este parámetro, se selecciona un valor medio entre los ya mencionados, es decir, relación longitud-diámetro de 15, por lo que el valor de la longitud es igual a 108cm. Sin embargo, debido a facilidad de manejo de valores, se decide otorgar un valor de 120cm, para el cual la relación longitud-diámetro es de 16,66, lo que significa que el valor sigue estando dentro del rango del parámetro anteriormente descrito y es un valor cercano, al inicialmente seleccionado.

[45]Fleeman. Eugene. Tactical Missile Design. American Institute of Aeronautics and Astronautics. 2001.

2.2.1. Ojiva

Una vez definidos los parámetros de la longitud y diámetros del fuselaje y con base en observaciones de las dimensiones de la ojiva en cohetes como ULA 1[46], ULA 2[47], Castor[48], cuyos valores oscilan entre el 13% y el 36% con respecto a la longitud del fuselaje del cohete; de acuerdo a los valores anteriores, se decide tomar como valor de la longitud de ojiva en el *ARISTARCO I, el 25% de* l_f, el cual corresponde a un valor de 30cm. Este valor es obtenido a través de la ecuación 3.

Ecuación 3. Longitud de la Ojiva

$$l_O = 0.25(l_F)$$

Fuente: Autores.

Reemplazando en la ecuación 3, se obtiene la longitud de la ojiva del *ARISTARCO I*:

$$l_O = 0.25(120cm)$$

$$l_O = 30cm$$

El valor de longitud de ojiva, anteriormente determinado tiene relación de longitud-diámetro de 4,16; el cual es cercano al parámetro de diseño considerado óptimo para ojivas, cuya relación longitud-diámetro, es de 5.[49]

2.2.2. Fuselaje

De acuerdo a lo anterior a lo anterior, el fuselaje posee una longitud de 120cm, a lo largo de la cual, se incorporarán el acople de la ojiva, el sistema de recuperación, la carga paga, aislantes térmicos, cámara de combustión y aletas estabilizadoras.

---

[46] Marcano. V, Benitez. P, La Rosa. C, La Cruz. L, Parco. M, Ferreira. J, Andressen. R, Serra. A, Peñaloza. M, Rodriguez. L, Cardenas. J, Minitti. V y Rojas. J. PROGRESOS ALCANZADOS EN EL PROYECTO UNIVERSITARIO COHETE SONDA ULA. Venezuela. 2009.
[47] Ibid.
[48] De Leon. Pablo. HISTORIA DE LA ACTIVIDAD ESPACIAL ARGENTINA. Argentina. 2008. p. 232.
[49] Fleeman. Eugene. Tactical Missile Design. American Institute of Aeronautics and Astronautics. 2001.

Mediante la ecuación 4, se obtiene el valor de la longitud total del cohete, con base en la longitud de la ojiva y del fuselaje.

Ecuación 4. Longitud Total del Cohete.

$$l_T = l_f + l_o$$

Fuente: Autores.

Reemplazando en la ecuación 4, se obtiene la longitud del ARISTARCO *I*:

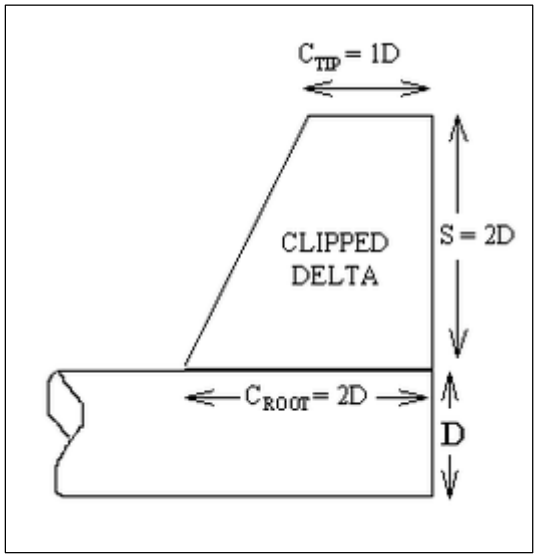
$$l_T = 120cm + 30cm$$

$$l_T = 150cm$$

2.2.3. Aletas

Las dimensiones de las aletas del cohete sonda *ARISTARCO I,* son obtenidas a partir de la relación de longitudes de cada una de las partes de la aleta (C_{Root}, C_{Tip}, S), con referencia al diámetro máximo del fuselaje, establecido por la Escuela Argentina de Modelismo Espacial.[50]Las dimensiones mencionadas se muestran en la imagen 19, y están dadas por las ecuaciones 5. 6 y 7.

Imagen 19 Dimensionamiento de una aleta estabilizadora.



Fuente: Cohetería experimental amateur CEA, Las aletas estabilizadoras [en línea].

[50] Coheteria experimental amateur CEA, Las aletas estabilizadoras. [En linea].

Ecuación 5. Longitud de la cuerda de raíz (Root).

$$C_{Root} = 2(\emptyset_{fe})$$

Fuente: Cohetería experimental amateur CEA, Las aletas estabilizadoras [en línea].

Ecuación 6. Longitud de la cuerda de punta (Tip)

$$C_{Tip} = \emptyset_{fe}$$

Fuente: Cohetería experimental amateur CEA, Las aletas estabilizadoras [en línea].

Ecuación 7. Longitud de borde de fuga (S)

$$S = 2(\emptyset_{fe})$$

Fuente: Cohetería experimental amateur CEA, Las aletas estabilizadoras [en línea].

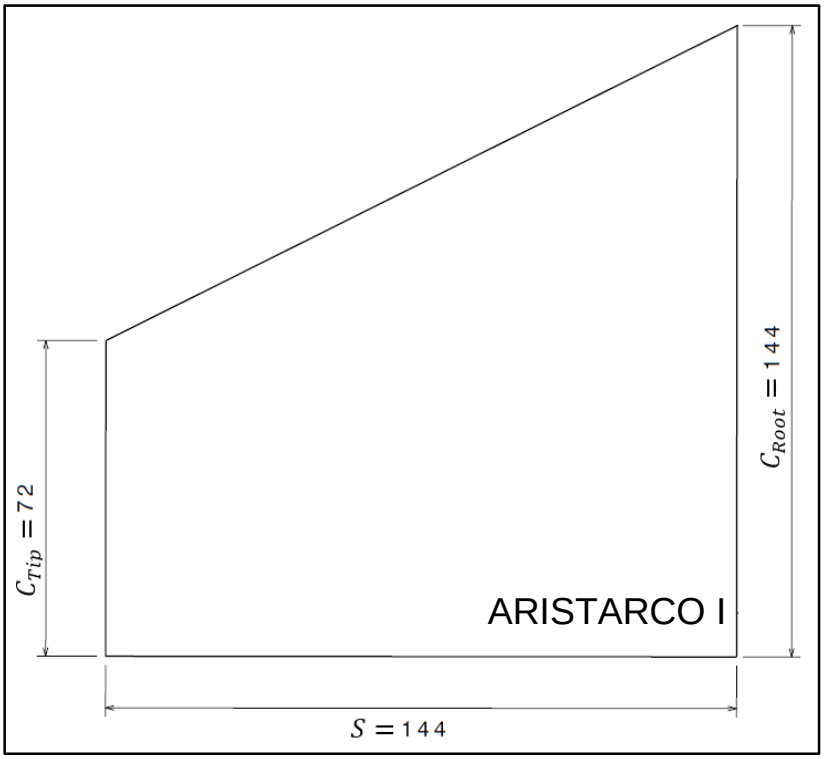
Reemplazando el valor del diámetro externo del fuselaje del cohete en las ecuaciones 5, 6 y 7, se obtienen los valores:

$$C_{Root} = 2(7{,}2cm) = 14{,}4cm$$

$$C_{Tip} = 7{,}2cm = 7{,}2cm$$

$$S = 2(7{,}2cm) = 14{,}4cm$$

Imagen 20. Dimensionamiento de la Aleta del ARISTARCO I (cotas en mm)



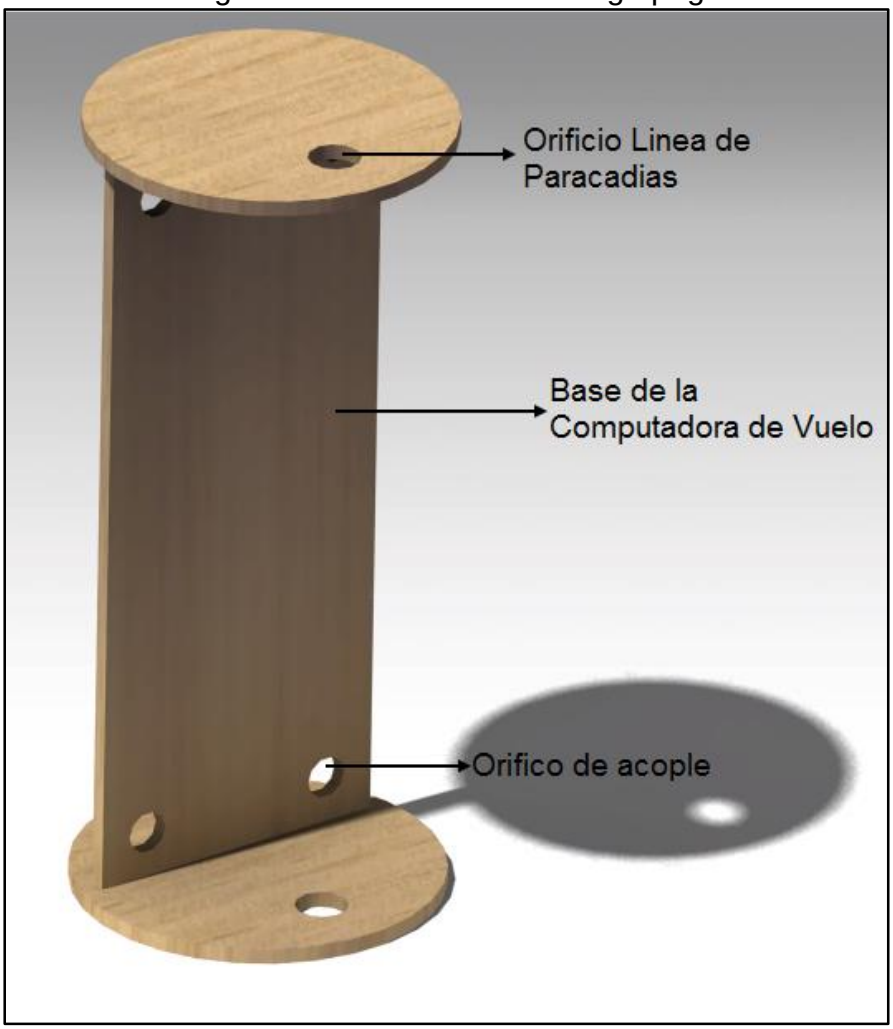
Fuente: Autores

2.2.4. Carga Paga

Las dimensiones de la Carga Paga del *ARISTARCO I,* están dadas por la configuración geométrica correspondiente a la computadora de vuelo, detallada en la sección 2.7. Dichas dimensiones son de aproximadamente 14,5cm de longitud y 6,5cm de ancho.

Debido a que se debe acoplar la geometría de la computadora de vuelo con el cuerpo cilíndrico del fuselaje del cohete, se opta por caracterizar una estructura rectangular con terminales circulares iguales al diámetro interior del cohete $(\text{Ver sección } 3.2.4)$ y longitud (l_{pl}) de 15cm, debido a las dimensiones de los componentes que integran la carga paga como la tarjeta ARDUINO, los sensores y el adaptador de tarjeta microSD (ver sección 2.7.), como se muestra en la Imagen 21.

Imagen 21. Estructura de la carga paga.



Fuente: Autores

2.2.5. Motor

Como se mencionó en la sección 1.5.5. el motor cohete está compuesto por tres partes.

2.2.5.1. Propelente

La longitud del propelente en la cámara de combustión, se determinó en 25cm (Ver sección 2.3.2.6.),el diámetro de este se estableció según la experiencia del semillero de cohetería de la Fundación Universitaria Los Libertadores (AERODES&I) en este campo, con el diámetro interno del material utilizado como aislante entre el propelente y la pared interior de la cámara de combustión, cuyos

diámetrosson 3,16cm y 3,5cm, respectivamente. Los diámetros anteriormente mencionados, son seleccionados con base en las dimensiones de los implementos que actualmente posee el semillero AERODES&I, para realizar la elaboración del propelente.[51]

2.2.5.2. Cámara de Combustión

La cámara de combustión del motor del cohete *ARISTARCO I,* se determinó, con base en los diámetros externo e interno del material disponible en existencia en el semillero de cohetería de la Fundación Universitaria Los Libertadores, cuyos valores corresponden a 3,8cm y 3,5cm, respectivamente.[52]Adicional a la disponibilidad del material, dichos diámetros son seleccionados debido a que son dimensiones acordes a las dimensiones del fuselaje, ya que permiten ser incorporados en el interior de este y son dimensiones existentes en cilindros del material de fácil acceso comercialmente. La longitud de este se determinó con base en la longitud del propelente (Sección 2.2.5.1.), ya que este componente es el encargado de contener el propelente, por lo tanto, la longitud de la Carama de Combustiónes de 25cm.

2.2.5.3. Tobera

Mediante los resultados obtenidos de los cálculos realizados en la sección 2.5. la tobera para el motor del *ARISTARCO I,* tiene las siguientes dimensiones: diámetro de entrada ($Ø_{Conv}$) de 3,5cm, diámetro de garganta ($Ø_G$) de 0,942cm, diámetro de salida ($Ø_{Div}$) de 6,036 cm, ángulo de convergencia (θ) de 45°, ángulo de divergencia (β) de 12° y longitud total de 13,559cm (Ver sección 2.5.).

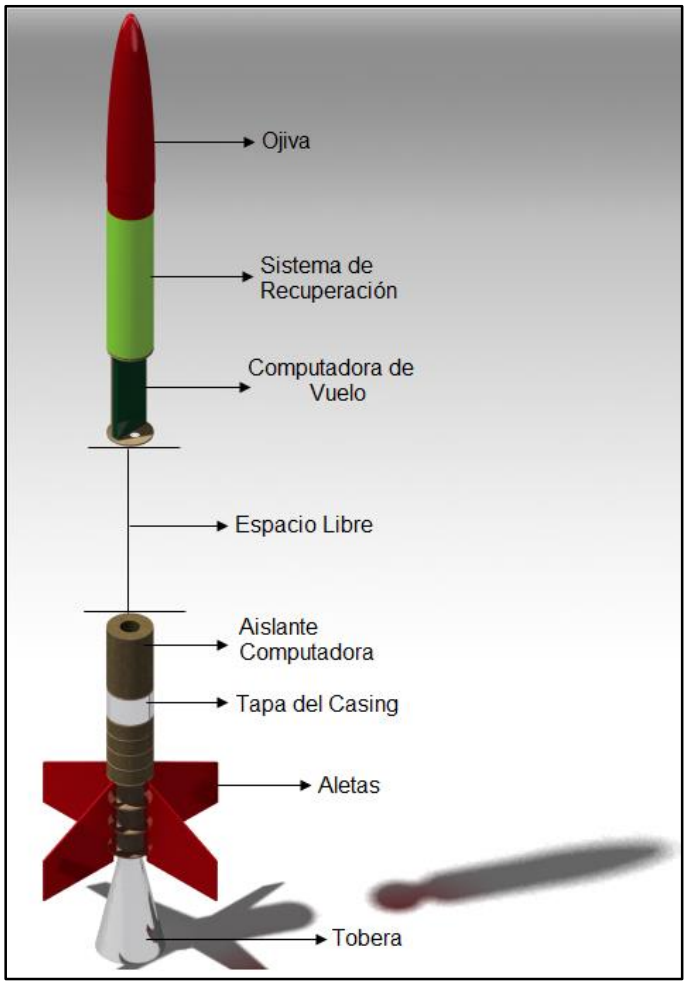
2.2.6. Distribución de los componentes en el cohete

En el interior del fuselaje del cohete, a 1cm desde la base del mismo, se encuentra posicionado el sistema de propulsión (SP) con una longitud de 27cm, ya que el sistema mencionado debe encontrarse en la parte inferior del vehículo, con fin de realizar la combustión de forma que no se arriesgue ningún elemento del cohete;

[51] Gomez, Fabian. Leiva, Huindi. ANÁLISIS DEL RENDIMIENTO DEL PROPELENTE SOLIDO TIPO AMATEUR MEDIANTE BALLISTIC EVALUATION MOTOR (BEM) Y SELECCIÓN DE LA TOBERA MAS ADECUADA PARA SU USO EN EL COHETE SONDA LIBERTADOR I. Bogotá DC. 2015.
[52] Ibid.

seguido de este, a 28cm de la base del vehículo, se encuentra ubicado el aislante térmico de la carga paga (AisPL) con una longitud de 10.5cm, cuyo fin es aislar térmicamente la cámara de combustión del resto de componentes del cohete; posterior a este, a 77cm de la base del *ARISTARCO I*, con longitud de 15cm se encuentra la carga paga (PL) y a continuación, es decir a 92cm de la base del cohete, está ubicado el sistema de recuperación (SR), con longitud de 25cm, ya que este debe encontrarse inmediatamente seguido de la ojiva, ya que es el encargado de expulsar esta y a su vez el paracaídas de recuperación (Ver Imagen 22).

Imagen 22. Ubicación de los componentes internos del ARISTARCO I



Fuente: Autores

2.3. Masa de Lanzamiento

Para hacer una aproximación de la masa del cohete *ARISTARCO I,* se realiza el cálculo la densidad de cada uno de los materiales de los componentes del cohete.

(Ojiva, fuselaje, aletas, Cámara de Combustión, cartón aislante y tobera), la cual se define como la cantidad de masa por unidad de volumen que ocupa un cuerpo determinado.[53]

Para determinar la densidad de cada uno de los elementos mencionados, se realizó la medición de la masa de componentes similares disponibles en el semillero de cohetería de la Fundación Universitaria Los Libertadores, posteriormente la determinación del volumen ocupado por cada uno de estos, y a través de estos se calcula la densidad de cada uno de ellos, mediante la ecuación 8.

Ecuación 8 Densidad de un material.

$$\rho = \frac{m}{V}$$

Fuente: Reboiras. M.QUIMICA: La ciencia básica. Editorial Thompson. 2008. p. 15.

Donde $m,$ es la masa del material y V, el volumen que ocupa el cuerpo.

Para determinar el volumen ocupado por los elementos, se opta por determinar el valor del volumen de la geometría externa de cada componente, denominado en este documento como volumen externo (V_e), posteriormente se calcula el volumen de las geometrías internas (aquellas que no están ocupadas por un cuerpo) de los componentes, denominado en este caso volumen interno (V_i); la diferencia existente entre el volumen externo y el interno, da como resultado el volumen que ocupa el cuerpo, denominado en este documento como volumen real (V_R).

2.3.1. Características de los elementos disponibles en el semillero

2.3.1.1. Ojiva

La ojiva utilizada para determinar sus características, es la que se encuentra disponible en el semillero de cohetería anteriormente descrito, está fabricada en fibra de vidrio, esta posee longitud (l_{o-I}) de 20cm, diámetro $(\emptyset_{o-I})$ de 7,2cm y espesor (e_{o-I}) de 0,5cm. La masa de esta es de 146gr.

El volumen de la ojiva está dado por la ecuación 9:

[53]Reboiras. M. QUIMICA: La ciencia básica. Editorial Thompson. 2008. p. 15.

Ecuación 9. Volumen de Ojiva Elíptica.

$$V_o = \frac{\pi(\emptyset_o)^2(l_o)}{6}$$

Fuente: NASA. Nose Cone Volume. [En línea]

Donde V_o es el volumen de la ojiva, $\emptyset_o$ el diámetro de la ojiva y l_o la longitud de la ojiva.

El volumen real que ocupa la ojiva, está dado por la ecuación 10.

Ecuación 10. Volumen Real de Ojiva.

$$V_{oR} = (V_{oe}) - (V_{oi})$$

Fuente: Autores

Donde V_{oR} es el volumen real de la ojiva, V_{oe}el volumen externo de la ojiva y V_{oi} el volumen interno de la misma.

Reemplazando en la ecuación 9los valores de la ojiva en cuestión, se obtienen los resultados de volumen externo (v_{oe-I}) y volumen interno (v_{oi-I}).

$$V_{oe-I} = \frac{\pi(7.2cm)^2(20cm)}{6} = 542{,}86cm^3$$

$$V_{oi-I} = \frac{\pi(7.2cm - 1cm)^2(20cm - 0{,}5cm)}{6} = 392{,}47917cm^3$$

Los valores obtenidos, son reemplazados en la ecuación 10.

$$V_{oR-I} = 542{,}86cm^3 - 392{,}47917cm^3 = 150{,}38804cm^3$$

Al reemplazar los valores de masa y volumen obtenidos en la ecuación 8, se determina el valor de la densidad que tiene la ojiva.

$$\rho_o = \frac{146\text{gr}}{150{,}38804 cm^3} = 0{,}97082188 \frac{\text{gr}}{cm^3}$$

2.3.1.2. Fuselaje

El diámetro externo de la sección fuselaje disponible en el semillero de cohetería, es de 7,2cm y el diámetro interno es de 6,935cm. La longitud de la muestra que se analizó, es de 42cm. El material en el cual está hecha esta sección, es fibra fenólica. La masa dicha sección analizada, es de 99gr.

Mediante la ecuación 11, se calcula el volumen de un cilindro.

Ecuación 11. Volumen de un Cilindro

$$V_{Cil} = \pi(r^2)(l)$$

Fuente: OnlineMSchool. Volumen del cilindro [en línea].

Donde V_{Cil} es el volumen de un cilindro, r es el radio de la base del cilindro y l es la longitud del mismo.

Ecuación 12. Volumen Real de Cilindro

$$V_{fR} = (V_{fe}) - (V_{fi})$$

Fuente: Autores.

Donde V_{fR} es el volumen real de la sección de fuselaje, V_{fe} es el volumen externo de la sección de fuselaje y V_{fi} el volumen interno de este.

Así, el volumen externo e interno del cilindro, están dados por la ecuación 11, al reemplazar los valores correspondientes a la sección de fuselaje analizada:

$$V_{fe-I} = \pi(3{,}6cm)^2(42cm) = 1710{,}03171 cm^3$$

$$V_{fi-I} = \pi(3{,}4675cm)^2(42cm) = 1586{,}47087cm^3$$

Al sustituir los valores de volumen exterior e interior en la ecuación 12, se obtiene el volumen real de la sección de fuselaje.

$$V_{fR} = 1710{,}03171cm^3 - 1586{,}47087cm^3 = 123{,}560842cm^3$$

Para determinar la densidad de este material, en la ecuación 8, se reemplazan los valores pertenecientes a la sección cilíndrica en cuestión.

$$\rho_f = \frac{99\text{gr}}{123{,}560842cm^3} = 0{,}80122471\frac{\text{gr}}{cm^3}$$

2.3.1.3. Aletas

Las aletas observadas poseen configuración de flechado positivo trapezoidal, cuyo material de fabricación es madera tipo triplex. Las geometrías de estas corresponden a trapecios rectangulares, cuyas dimensiones son: C_{Root} de 9,61cm, S de 8,47cm y C_{Tip} de 4,27cm. El espesor e_A de las aletas mencionadas, es de0,33cm. La masa de una de las aletas estabilizadoras mencionadas, es de 18gr.

La superficie de una aleta estabilizadora trapezoidal, es igual al área de un trapecio rectángulo, la cual está dada por la ecuación 13.

Ecuación 13. Superficie de Aletas Trapezoidales.

$$A_A = S\frac{(C_{Root} + C_{Tip})}{2}$$

Fuente: Calcular Área. Calculo de la Superficie del Trapecio [en línea].

Al reemplazar los valores en la ecuación 13, se obtiene el valor del área mencionada.

$$A_{A-I} = 8{,}47cm\frac{(9{,}61cm + 4{,}27cm)}{2} = 58{,}782cm^2$$

El volumen de un cuerpo prismático como lo es una aleta estabilizadora en este caso, está dado por la ecuación 14.

Ecuación 14. Volumen de un Cuerpo Prismático.

$$V = A_b(h)$$

Fuente: OnlineMSchool. Volumen de la Prisma [en línea].

Donde V es el volumen del cuerpo, A_b el área de la base del cuerpo y h es la altura del mismo.

Reemplazando los valores del área de la base (A_{A-I}) y altura del cuerpo (e_A), para el caso de las aletas estabilizadoras en la ecuación 14, se obtiene:

$$V_{A-I} = 58{,}782cm^2(0{,}33\text{cm}) = \ 19{,}39cm^3$$

Mediante la ecuación 8, se determina la densidad del material.

$$\rho_A = \frac{18gr}{19{,}39cm^3} = 0{,}92831356\frac{gr}{cm^3}$$

2.3.1.4. Cámara de Combustión

El material en el cual está fabricado el Cámara de Combustión del semillero de cohetería es Aluminio 7075, cuyas dimensiones son: diámetro exterior de 3,8cm, diámetro interior de 3,5cm, longitud de 23,5cm y masa de 102gr.

Reemplazando los valores correspondientes para este cilindro, las ecuaciones 11 y 12 permiten determinar el volumen que ocupa este cuerpo.

$$V_{\boldsymbol{Case-I}} = \pi(1{,}9cm)^2(23{,}5cm) = 266{,}517013cm^3$$

$$V_{\text{Casi}-I} = \pi(1{,}75cm)^2(23{,}5cm) = 226{,}096496cm^3$$

$$V_{CasR} = 266{,}517013cm^3 - 226{,}096496cm^3 = 40{,}4205165cm^3$$

Mediante la ecuación 8, se obtiene el valor de la densidad del material en cuestión.

$$\rho_{Cas} = \frac{102\text{gr}}{40{,}4205165 cm^3} = 2{,}52347097 \frac{\text{gr}}{cm^3}$$

2.3.1.5. Aislante de la Cámara de Combustión

El material para aislar el propelente de la pared interior de la cámara de combustión, utilizado en el semillero de cohetería anteriormente descrito, es cartón cilíndrico con diámetro exterior de 3,5cm, diámetro interno de 3,16cm, longitud de 5cm y masa de 6gr.

Reemplazando los valores correspondientes para este cilindro, las ecuaciones 11 y 12 permiten determinar el volumen que ocupa este cuerpo.

$$V_{aisce-I} = \pi(1{,}75cm)^2(5cm) = 48{,}931201 cm^3$$

$$V_{aisci-I} = \pi(1{,}58cm)^2(5cm) = 39{,}8407733 cm^3$$

$$V_{aiscR} = 48{,}931201 cm^3 - 39{,}8407733 cm^3 = 9{,}09042793 cm^3$$

Mediante la ecuación 8, se obtiene el valor de la densidad del material en cuestión.

$$\rho_{aisc} = \frac{6\text{gr}}{9{,}09042793 cm^3} = 0{,}66003493 \frac{\text{gr}}{cm^3}$$

2.3.1.6. Propelente

El propelente con el que cuenta el semillero de cohetería, es propelente solido amateur tipo Candy, el cual, ha sido utilizado en pruebas de motor cohete con dimensiones de: diámetro exterior de 3,16cm, longitud de 5cm y masa de 74gr.[54]

Reemplazando los valores correspondientes para este cilindro, la ecuación 11 permite determinar el volumen que ocupa este cuerpo.

$$V_{\mathrm{PropR}} = \pi(1{,}58cm)^2(5cm) = 39{,}84077325cm^3$$

Mediante la ecuación 8, se obtiene el valor de la densidad del material en cuestión.

$$\rho_{Prop} = \frac{74\mathrm{gr}}{39{,}84077325cm^3} = 1{,}85739367\frac{\mathrm{gr}}{cm^3}$$

Los datos de las características de los materiales en los cuales se fabrican algunas secciones del cohete Ojiva, Fuselaje, Aletas, Cámara de Combustión del motor, Aislante de la cámara de combustión y Propelente), obtenidos en la sección 2.3.1. se presentan en la tabla 1.

Tabla 1 Resultados de las características de los materiales disponibles en semillero de cohetería

Componente	**MASA (gr)**	**VOLUMEN** $(\boldsymbol{cm^3})$	**DENSIDAD** $\left(\frac{\mathbf{gr}}{\boldsymbol{cm^3}}\right)$
Ojiva	146	150,38804	0,97082188
Fuselaje	99	123,560842	0,80122471
Aletas Estabilizadoras	18	19,39	0,92831356
Cámara de Combustión	102	40,4205165	2,52347097
Aislante *Cámara de Combustión*	6	9,09042793	0,66003493
Propelente	74	39,8407733	1,85739367

Fuente: Autores.

[54] Gomez, Fabian. Leiva, Huindi. ANÁLISIS DEL RENDIMIENTO DEL PROPELENTE SOLIDO TIPO AMATEUR MEDIANTE BALLISTIC EVALUATION MOTOR (BEM) Y SELECCIÓN DE LA TOBERA MAS ADECUADA PARA SU USO EN EL COHETE SONDA LIBERTADOR I. Bogota DC. 2015.

2.3.2. Características del cohete ARISTARCO I

2.3.2.1. Ojiva

La ojiva del cohete sonda *ARISTARCO I*, será fabricada en fibra de vidrio, esta posee las siguientes dimensiones: longitud (l_o) de 30cm, diámetro $(\emptyset_o)$ de 7,2cm y en espesor (e_o) de 0,5cm.

Reemplazando en la ecuación 9 los valores de la ojiva en cuestión, se obtienen los resultados de volumen externo (v_{o-Iext}) y volumen interno (v_{o-Iint}).

$$V_{oe} = \frac{\pi(7.2cm)^2(30cm)}{6} = 814{,}300816cm^3$$

$$V_{oi} = \frac{\pi(7.2cm - 1cm)^2(30cm - 0{,}5cm)}{6} = 593{,}75054cm^3$$

Los valores obtenidos, son reemplazados en la ecuación 10.

$$V_{oR} = 814{,}300816cm^3 - 593{,}75054cm^3 = 220{,}550276cm^3$$

2.3.2.2. Fuselaje

Para el cohete *ARISTARCO I,* el diámetro externo del fuselaje es de 7,2cm y el diámetro interno es de 6,935cm. La longitud es de 120cm. El material en el cual será realizada esta sección, es fibra fenólica.

Así, el volumen externo e interno del cilindro, están dados por la ecuación 11, al reemplazar los valores correspondientes a la sección de fuselaje se tiene:

$$V_{fe} = \pi(3{,}6cm)^2(120cm) = 4885{,}80489cm^3$$

$$V_{fi} = \pi(3{,}4675cm)^2(120cm) = 4532{,}77392cm^3$$

Al reemplazar los valores de volumen exterior e interior en la ecuación 12, se obtiene el volumen real de la sección de fuselaje.

$$V_{fR} = 4885{,}80489cm^3 - 4532{,}77392cm^3 = 353{,}030977cm^3$$

2.3.2.3. Aletas

Como se describió en la sección 1.5.3. del presente documento, la configuración de las aletas estabilizadoras dispuestas para el cohete *ARISTARCO I* es de flechado positivo trapezoidal, cuyo material de fabricación seleccionado es madera tipo triplex. Las geometrías de estas corresponden a trapecios rectangulares, cuyas dimensiones son: C_{Root} de 14,4cm, S de 14,4cm y C_{Tip} de 7,2cm. El espesor e_A de las aletas mencionadas, es de0,3cm.El espesor de estas, influye en la rigidez, masa y superficie que tengan las mismas. El valor seleccionado es debido a las características observadas de aletas estabilizadoras realizadas en el semillero AERODES&I y la existencia comercial de este espesor en el material seleccionado.

La superficie de una aleta estabilizadora trapezoidal, es igual al área de un trapecio rectángulo, la cual está dada por la ecuación 13.

Al reemplazar los valores en la ecuación 13, se obtiene el valor del área mencionada.

$$A_A = 14{,}4cm\frac{(14{,}4cm + 7{,}2cm)}{2} = 155{,}52cm^2$$

Reemplazando los valores del área de la base (A_A) y altura del cuerpo (e_A), para el caso de las aletas estabilizadoras en la ecuación 14, se obtiene:

$$V_A = 155{,}52cm^2(0{,}3\text{cm}) = 46.656cm^3$$

2.3.2.4. Cámara de Combustión

El material seleccionado para la fabricación del Cámara de Combustión del motor del cohete *ARISTARCO I* es Aluminio 7075, cuyas dimensiones son: diámetro exterior de 3,8cm, diámetro interior de 3,5cm y longitud de 25cm (Ver sección 2.3.2.6.).

Reemplazando los valores correspondientes para este cilindro, las ecuaciones 11 y 12 permiten determinar el volumen que ocupa este cuerpo.

$$V_{\boldsymbol{Case}} = \pi(1{,}9cm)^2(25cm) = 283{,}528cm^3$$

$$V_{\text{Casi}} = \pi(1{,}75cm)^2(25cm) = 240{,}528cm^3$$

$$V_{CasR} = 283{,}528cm^3 - 240{,}528cm^3 = 43cm^3$$

2.3.2.5. Aislante de la Cámara de Combustión

El material para aislar el propelente de la pared interior de la cámara de combustión, seleccionado para implementar en el motor del *ARISTARCO I,* es fibra fenólica, de geometría cilíndrica con diámetro exterior de 3,5cm, diámetro interno de 3,16cm y longitud de 25cm (Ver sección 2.3.2.6.).

Reemplazando los valores correspondientes para este cilindro, las ecuaciones 11 y 12 permiten determinar el volumen que ocupa este cuerpo.

$$V_{aisce} = \pi(1{,}75cm)^2(25cm) = 240{,}528cm^3$$

$$V_{aisci} = \pi(1{,}58cm)^2(25cm) = 196{,}066cm^3$$

$$V_{aiscR} = 240{,}528cm^3 - 196{,}066cm^3 = 44{,}461cm^3$$

2.3.2.6. Propelente

El propelente con el que se decidió trabajar para el desarrollo del *ARISTARCO I,* es el mismo propelente con el que cuenta el semillero de cohetería, el cual es propelente solido amateur tipo Candy, cuyas dimensiones son: diámetro exterior de 3,16cm. Una vez se estableció la longitud de los componentes O (5cm) SR (25cm), PL (15cm) y AisPL (10.5cm), en el interior del fuselaje, se concluye que para el espacio del sistema de propulsión quedan 61cm, de los cuales se determinan 25cm que ocupara el sistema de propulsión, debido a que como se describe en la sección 1.2. la relación de masa del vehículo, es recomendable que tenga un valor cercano a 1 (Ver sección 2.4.).

Reemplazando los valores correspondientes para este cilindro, la ecuación 11 permite determinar el volumen que ocupa este cuerpo.

$$V_{\mathrm{PropR}} = \pi(1{,}58cm)^2(25cm) = 196{,}066cm^3$$

Los datos de las características de algunas secciones del cohete (Ojiva, Fuselaje, Aletas, Cámara de Combustión del motor, Aislante de la cámara de combustión y Propelente), obtenidos en la sección 2.3.2. se presentan en la tabla 2.

Tabla 2. Resultados de las características de los componentes evaluados en la sección 2.3.2.

COMPONENTE	VOLUMEN($\boldsymbol{cm^3}$)
Ojiva	220,550276
Fuselaje	353,030977
Aletas Estabilizadoras	46,656
Cámara de Combustión	43
Aislante Cámara de Combustión	44,431
Propelente	196,066

Fuente: Autores.

2.3.3. Estimación de la Masa del Cohete ARISTARCO I

De la ecuación 8, al despejar la variable masa, y con base en los datos registrados en las tablas 1 y 2, se obtiene el valor de la masa estimada de los componentes descritos en la sección 2.3.2.

Así de la ecuación 8, despejando la variable masa, la ecuación queda como se expresa en la ecuación 15.

Ecuación 15. Masa de un Componente.

$$m = (\rho)(V)$$

Fuente: Reboiras. M.QUIMICA: La ciencia básica. Editorial Thompson. 2008. p. 15.

Donde m es la masa determinada, ρ es la densidad del elemento y V es volumen que del cuerpo.

En la tabla 3, se presentan los datos de densidad de cada uno de los materiales descritos en la sección 2.3.1. y el volumen de los componentes descritos en la sección 2.3.2., cuyos valores están registrados en las tablas 2 y 3, respectivamente.

Tabla 3. Densidad y volumen y de los componentes del cohete ARISTARCO I.

COMPONENTE	**DENSIDAD** $(\frac{\mathbf{gr}}{\mathbf{cm^3}})$	**VOLUMEN** $(\boldsymbol{cm^3})$
Ojiva	0,97082188	220,550276
Fuselaje	0,80122471	353,030977
Aletas Estabilizadoras	0,92831356	46,656
Cámara de Combustión	2,52347097	43
Aislante Cámara de Combustión	0,66003493	44,431
Propelente	1,85739367	196,066

Fuente: Autores.

2.3.3.1. Ojiva

Al reemplazar en la ecuación 15 los valores correspondientes a la ojiva del cohete, registrados en la tabla 3, se obtiene la masa del componente en cuestión.

$$m_O = \left(0{,}97082188 \frac{\text{gr}}{cm^3}\right)(220{,}550276\ cm^3) = 214{,}115034\text{gr}$$

2.3.3.2. Fuselaje

Al reemplazar en la ecuación 15 los valores correspondientes al fuselaje del cohete, registrados en la tabla 3, se obtiene la masa del componente en cuestión.

$$m_F = \left(0{,}80122471 \frac{\text{gr}}{cm^3}\right)(353{,}030977\ cm^3) = 282{,}857143\text{gr}$$

2.3.3.3. Aletas

Al reemplazar en la ecuación 15 los valores correspondientes a las aletas estabilizadoras del cohete, registrados en la tabla 3, se obtiene la masa del componente en cuestión.

$$m_A = \left(0{,}92831356 \frac{\text{gr}}{cm^3}\right)(46{,}656\ cm^3) = 173{,}245591\text{gr}$$

2.3.3.4. Aislante Carga Paga

En la sección 1.7 del documento, se describe el aislante que se decide utilizar para aislar térmicamente la carga paga del motor del cohete.

La longitud de este se establece en 10,5cm, y su diámetro igual al diámetro interno del fuselaje del vehículo (6,935cm).

Según la experiencia adquirida por los autores en diversas actividades de manipulación de dicho material se obtiene que el valor de la masa para las dimensiones mencionadas es de 56,659 gr

2.3.3.5. Carga Paga

Como se especifica en la sección 1.9. la masa de la computadora de vuelo con sensores de medición (PL) utilizada, es de 51gr. Adicionando la masa aproximada de la estructura que la sostiene en el interior del fuselaje, cuyo valor es de aproximadamente 49gr, el valor de la masa del componente PL, es de 100gr.

2.3.3.6. Sistema de Recuperación

El sistema de recuperación estipulado para ser utilizado en el *ARISTARCO I,* es el que actualmente posee el semillero de cohetería de La Fundación Universitaria Los Libertadores. El valor de este sistema es de 250gr, incluyendo su estructura de sujeción y mecanismo de eyección. Su longitud es de 25cm.

2.3.3.7. Sistema de Propulsión

Como se han mencionado anteriormente el Sistema de Propulsión (SP) de divide en 4 partes, a continuación, se encuentra la estimación de masa para cada uno de estos componentes.

2.3.3.7.1. Cámara de Combustión

Al reemplazar en la ecuación 15 los valores correspondientes a laCámara de Combustión del motor del cohete, registrados en la tabla 3, se obtiene la masa del componente en cuestión.

$$m_{Cas} = \left(2{,}52347097 \frac{\text{gr}}{cm^3}\right)(43cm^3) = 108{,}5106\text{gr}$$

2.3.3.7.2. Aislante de Cámara

Al reemplazar en la ecuación 15 los valores correspondientes al aislante de la cámara de combustión del motor del cohete, registrados en la tabla 3, se obtiene la masa del componente en cuestión.

$$m_{AisC} = \left(0{,}66003493 \frac{\text{gr}}{cm^3}\right)(44{,}461 cm^3) = 29{,}34\text{gr}$$

2.3.3.7.3. Propelente

Al reemplazar en la ecuación 15 los valores correspondientes al propelente del motor del cohete, registrados en la tabla 3, se obtiene la masa del componente en cuestión.

$$m_p = \left(1{,}85739367 \frac{\text{gr}}{cm^3}\right)(196{,}066\ cm^3) = 364{,}17\text{gr}$$

2.3.3.7.4. Tobera

Según la experiencia de los autores en toberas previamente realizadas, la masa de una tobera es de aproximadamente 200gr, valor que se toma para la tobera del *ARISTARCO I.*

2.3.3.8. Masa Total del Cohete

Con base en los resultados obtenidos de la masa de cada uno de los componentes del ARISTARCO I, en la tabla 4 se muestran los resultados ya mencionados y la sumatoria de estos, la cual determina la masa total de lanzamiento del cohete ARISTARCO I.

Tabla 4. Masa de cada componente del ARISTARCO I y masa total del mismo.

ARISTARCO I	
COMPONENTE	**MASA (gr)**
Ojiva	214,115034
Fuselaje	282,857143
Aletas	173,245591
Aislante PL	56
Carga Paga	100
Sistema de Recuperación	250
Cámara de Combustión	108,5106
Aislante Cas	29,34
Propelente	364,17
Tobera	200
Masa Total	**1778,248**

Fuente: Autores.

De esta manera la masa total del cohete compuesta por la suma de la masa de cada uno de sus componentes, es de 1778,248gr.

2.3.3.9. Centro de Presiones

El centro de presiones es el lugar donde se concentran todas las fuerzas aerodinámicas que actúan sobre el cohete durante su vuelo.[55]

Durante el vuelo, las presiones generadas por las fuerzas aerodinámicas actuantes, serán ejercidas sobre cada uno de los componentes del vehículo que estén expuestos al flujo de aire (ojiva, fuselaje, aletas y sección divergente de tobera), en el punto del centro geométrico de cada uno de estos. De este modo, la fuerza actuara sobre la superficie del componente, generando una presión sobre este. Sin embargo, existe un punto donde la distribución de presiones que estén por delante de este punto, será igual a la distribución de presiones que se encuentre por detrás del mismo.[56]

[55] Recuenco. Jesus. MANUAL DEL CONSTRUCTOR DE MODELOS ESPACIALES. 2008. p. 17.
[56] NASA. Determining Center of Pressure-cp. [En línea].

A partir de lo anterior, la distancia del centro de presiones del vehículo, se puede determinar mediante la ecuación 16.

Ecuación 16. Determinación de la Distancia de CP en el Cohete.

$$(d_{CP}) = \frac{((d_O)(A_O) + (d_F)(A_F) + (d_A)(A_A) + (d_{To})(A_{To}))}{(A_{Tot})}$$

Fuente: NASA. Determining Center of Pressure-cp.[En linea]

Donde d_{CP} es la distancia de una línea de referencia al CP del vehículo, d_O la distancia de la línea de referencia al centro geométrico de la ojiva, d_F la distancia entre la línea de referencia y el centro geométrico del fuselaje, d_A la distancia de la línea de referencia hasta el centro geométrico de las aletas, d_{TO} la distancia desde la línea de referencia hasta el centro geométrico del área expuesta al flujo de aire de la tobera, A_O el área expuesta de la ojiva, A_F el área expuesta del fuselaje, A_A el área expuesta de las aletas, A_T el área expuesta de la tobera y A_{Tot} la sumatoria de las áreas de los componentes mencionados.

2.3.3.9.1. Centros Geométricos de los Componentes Expuestos

Debido a que el vehículo diseñado en el presente documento es simétrico sobre su eje longitudinal, el centro geométrico únicamente tiene valor diferente de cero sobre el eje longitudinal del vehículo mencionado.

2.3.3.9.1.1. Ojiva

La ojiva del cohete *ARISTARCO I,* con base en las dimensiones establecidas, posee una superficie expuesta de 450cm^2; dicho valor fue determinado mediante el software utilizado para realizar el diseño asistido por computador-CAD (CATIA).

El centro geométrico de una ojiva elíptica, está dado por el centro geométrico de media elipse, obtenido por la ecuación 17.

Ecuación 17. Centro Geométrico de Ojiva.

$$\bar{X}_O = \frac{4(l_O)}{3\pi}$$

Fuente: Inés Cedeño-Fisca. Área momento de inercia [en línea].

Reemplazando en la ecuación 17, se obtiene el valor del centro geométrico de la ojiva en el eje longitudinal desde la base de esta.

$$\bar{X}_O = \frac{4(30cm)}{3\pi} = 12{,}732cm$$

2.3.3.9.1.2. Fuselaje

La superficie expuesta del fuselaje del *ARISTARCO I,* está dado por la superficie lateral de un cilindro, la cual se determina mediante la ecuación 18.

Ecuación 18. Área Lateral de Fuselaje.

$$A_{latF} = 2\pi(r_F)(l_F)$$

Fuente: OnlineMSchool. Área de Cilindro [en línea].

Reemplazando los valores pertinentes en la ecuación 18, se obtiene:

$$A_{latF} = 2\pi(7{,}2cm)(120cm) = 5428{,}67cm^2$$

El centro geométrico del fuselaje, está dado por el centro geométrico de un cilindro, el cual es obtenido por laecuación 19.

Ecuación 19. Centro Geométrico de Fuselaje.

$$\bar{X}_F = \frac{l_F}{2}$$

Fuente: Física Básica. Centro de Gravedad [en línea].

Reemplazando en la ecuación 19, se obtiene el valor del centro geométrico del fuselaje en el eje longitudinal desde la base de este.

$$\bar{X}_F = \frac{120}{2} = 60cm$$

2.3.3.9.1.3. Aletas Estabilizadoras

Una aleta estabilizadora del cohete *ARISTARCO I,* con base en las dimensiones establecidas, posee una superficie expuesta de 178,125cm^2; dicho valor, equivale a una sola cara de la aleta; este fue determinado mediante el software utilizado para realizar el diseño asistido por computador-CAD (CATIA).

Teniendo en cuenta que el vehículo lleva cuatro aletas estabilizadoras, y a que cada aleta posee Extradós e Intradós expuestos al flujo de aire, el valor anteriormente obtenido, es necesario multiplicarlo por ocho, como se evidencia en la ecuación 20.

Ecuación 20. Superficie de Cuatro Aletas Estabilizadoras.

$$A_{4.A} = 8(A_A)$$

Fuente: Autores.

Así, reemplazando en la ecuación 20, se obtiene el valor de las cuatro aletas que posee el cohete.

$$A_{4.A} = 8(178{,}125cm^2) = 1425cm^2$$

El valor del centro geométrico es de 5,6cm sobre el eje longitudinal del cohete, contando desde la parte posterior de las aletas; dicho valor fue determinado mediante el software de construcción del modelo (CATIA).

2.3.3.9.1.4. Sección Divergente de Tobera

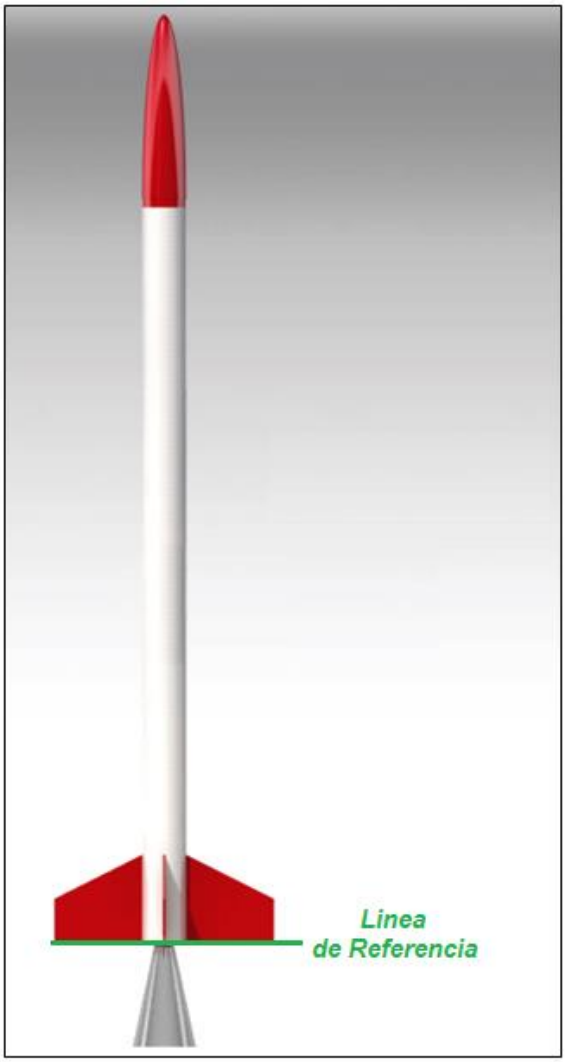
La sección divergente de la tobera del cohete *ARISTARCO I,* con base en las dimensiones establecidas, posee una superficie expuesta de 320,24cm^2; dicho valor fue determinado mediante el software utilizado para realizar el diseño asistido por computador-CAD (CATIA).

El valor del centro geométrico es de 5,25cm sobre el eje longitudinal del cohete, desde la sección de garganta de la tobera; dicho valor fue determinado mediante el software de construcción del modelo (CATIA).

2.3.3.9.2. Determinación del Centro de Presiones

Para determinar el punto en el que se encuentra el centro de presiones, se establece fijar una línea de referencia (LR) a partir de la cual son tomadas la distancia del centro geométrico de cada uno de los componentes, en el extremo posterior del cohete, como se muestra en la imagen 23.

Imagen 23. Ubicación de la Línea de Referencia cohete ARISTARCO I.



Fuente: Autores

En la tabla 5, se relacionan los datos obtenidos en la sección 2.3.3.10.1. y la distancia de los centros geométricos de los componentes respecto a la línea de referencia.

Tabla 5. Centro Geométrico, Distancia de Centro Geométrico desde LR y Área de Componentes Expuestos al Flujo.

Componente	$\bar{X}_n$ **(cm)**	d_n $(\mathbf{cm})$	A_n (cm^2)
Ojiva	12,732	132,732	450
Fuselaje	60	60	5428,67
Aletas Estabilizadoras	5,6	5,6	1425
Sección divergente de tobera.	11,136	-5,25	320,24
SUMATORIA	-	-	7623,67

Fuente: Autores

Reemplazando los datos de la tabla 6, en la ecuación 16, se obtiene el punto con referencia a RL en el que se encuentra el centro de presiones (CP).

$$(\mathrm{d_{CP}}) = \frac{((132{,}732\mathrm{cm})(450\mathrm{cm^2}) + (60\mathrm{cm})(5428{,}67\mathrm{cm^2}) + (5{,}6\mathrm{cm})(1425\mathrm{cm^2}) + (-5{,}25\mathrm{cm})(320{,}24\mathrm{cm^2}))}{(7623{,}67\mathrm{cm^2})}$$

$$(d_{CP}) = 51{,}385cm$$

2.3.3.10. Centro de Gravedad

Del mismo modo que el CP es el punto donde se concentran todas las fuerzas aerodinámicas actuantes sobre el vehículo, el CG es el punto donde se concentra todo el peso del cohete, lo que significa que el peso distribuido por delante de dicho punto es igual al peso distribuido detrás del mismo.[57]

De forma similar a la que actúa el CP, sobre los componentes del vehículo expuestos al flujo de aire durante el vuelo, el CG actúa sobre todos los

[57] Recuenco. Op.Cit., p.17.

componentes del cohete (O, F, A, SR, PL, SP, AisPL) con la masa correspondiente a cada uno de estos sobre su centro geométrico.[58]

De esta forma, con base en la ecuación 21, se determina el punto CG, con referencia a la LR seleccionada en la sección 2.3.3.10.2.

Ecuación 21. Determinaciónde la Distancia de CG en el Cohete.

$$(d_{CG}) = \frac{((d_O)(m_O) + (d_F)(m_F) + (d_A)(m_A) + (d_{SR})(m_{SR}) + (d_{PL})(m_{PL}) + (d_{SP})(m_{SP}) + (d_{AisPL})(m_{AisPL}))}{(m_{Tot})}$$

Fuente: NASA. Determining Center of Gravity-cg. [En linea].

2.3.3.10.1. Centros Geométricos

Anteriormente, la masa de cada componente del cohete fue determinada (ver tabla 5), así como el centro geométrico de algunos componentes (O, F y A), mostrados en la tabla 6.

2.3.3.10.1.1. Sistema de Recuperación

El sistema de recuperación del *ARISTARCO I*, como se establece en la sección 2.2.6. posee longitud de 25cm. Dado que este sistema se acopla al cuerpo cilíndrico del fuselaje, SR adquiere la misma geometría, por lo tanto, a través de la ecuación 19, se obtiene el valor de su centro geométrico.

$$\bar{X}_{SR} = \frac{25cm}{2} = 12{,}5cm$$

2.3.3.10.1.2. Carga Paga

Como se establece en la sección 2.2.6. del presente documento, la longitud de PL es de 15cm.

Ya que PL presenta geometría rectangular, el centro geométrico es determinado por la ecuación 22.

[58] NASA. Determining Center of Gravity-cg. [En línea].

Ecuación 22. Centro Geométrico de Carga Paga.

$$\bar{X}_{PL} = \frac{l_{PL}}{2}$$

Fuente: Inés Cedeño-Física. Área momento de inercia.

Reemplazando en la ecuación 22:

$$\bar{X}_{PL} = \frac{15cm}{2} = 7{,}5cm$$

2.3.3.10.1.3. Sistema de Propulsión

El sistema de propulsión, del *ARISTARCO I*, en esta sección, solo se tendrá en cuenta como conjunto de cámara de combustión-aislante de propelente-propelente, debido a que la tobera posee configuración geométrica diferente al conjunto mencionado. Como se establece en la sección 2.2.6. el conjunto mencionado, posee longitud de 25cm. Dado que este sistema posee cuerpo cilíndrico, a través de la ecuación 19, se obtiene el valor de su centro geométrico.

$$\bar{X}_{SP} = \frac{25cm}{2} = 12{,}5cm$$

2.3.3.10.1.4. Tobera

Mediante el software utilizado para realizar el modelo en CAD (CATIA), se determina que el centro geométrico de la tobera se encuentra a 2,15cm de la sección de entrada de esta. Esto se debe a la distribución de material, la cual es mucho mayor en la sección convergente. Dicho punto se encuentra muy próximo a la línea de referencia tomada, por lo que el valor para d_{To} se toma como cero.

2.3.3.10.2. Determinación del Centro de Gravedad

Para determinar el punto en el que se encuentra el centro de gravedad, se establece fijar una línea de referencia (LR) a partir de la cual son tomadas la distancia del centro geométrico de cada uno de los componentes, en el extremo posterior del cohete. (Ver sección 2.3.3.9.2)

En la tabla 6, se relacionan los datos obtenidos en la sección 2.3.3.10.1. en la sección 2.3.3.11.1. en la sección 2.3.3.9. y la distancia de los centros geométricos de los componentes respecto a la línea de referencia.

Tabla 6. Valores de Centro Geométrico, Distancia de Centro Geométrico desde LR y Masa de los Componentes del Cohete.

ITEM	$\bar{X}_n$ **(cm)**	d_n **(cm)**	m_n **(gr)**	$(d_n)(m_n)$ **(cm-gr)**
Ojiva	12,732	132,732	214,115	28419,9122
Fuselaje	60	60	282,8571	16971,426
Aletas Estabilizadoras	5,6	5,6	173,2456	970,17536
Sistema de Recuperación	12,5	104,5	250	26125
Carga Paga	7,5	84,5	100	8450
Sistema de Propulsión	12,5	13,5	502,029937	6777,40415
Aislante Carga Paga	7,5	69,5	56	3892
Tobera	2,05	0	200	0
SUMATORIA	-	-	1778,24764	91605,9177

Fuente: Autores.

Reemplazando los valores en la ecuación 21, se obtiene el valor de la distancia a la cual se encuentra el CG de LR.

$$(d_{CG}) = \frac{(28419{,}917\text{cmgr} + 16971{,}4286\text{cmgr} + 970{,}175\text{cmgr} + 26125\text{cmgr} + 8450\text{cmgr} + 6777{,}404\text{cmgr} + 3892\text{cmgr})}{(1778{,}247gr)}$$

$$(d_{CG}) = 51{,}515\text{cm}$$

De esta forma se obtiene que el CG está posicionado a 51,515cm de la línea de referencia y por lo tanto, el CG queda por delante del CP.

La estabilidad es entendida como una propiedad del sistema que en presencia de una perturbación no pierde su correcto comportamiento. La estabilidad se relaciona con la facilidad con la cual un cohete logra mantener la trayectoria y para

lograr este objetivo se debe garantizar que el centro de presiones (CP) permanezca por debajo del centro de gravedad (CG)[59].

El margen de estabilidad de un cohete, está definido por la diferencia que existe entre la posición del centro de gravedad y el centro de presiones.[60]

Ecuación 23. Margen de Estabilidad del Cohete

$$ME = d_{CG} - d_{CP}$$

Fuente: Recuenco. Jesús. MANUAL DEL CONSTRUCTOR DE MODELOS ESPACIALES. 2008. p.17.

Al reemplazar en la ecuación 23, se obtiene que el margen de estabilidad del *ARISTARCO I*.

$$ME = 51{,}515 - 51{,}385cm = 0{,}13cm$$

Debido a que la masa de propelente y de aislante de propelente, son los únicos parámetros de masa que varían con el paso del tiempo, el centro de gravedad también lo hará. Una vez se haya consumido la totalidad del propelente, el centro de gravedad quedara a 62,317cm de la línea de referencia. Lo que significa que al finalizar la combustión, el margen de estabilidad, será de 10,932cm.

2.4. Relación de Masa

Como fue definida en la sección 1.2. la relación de masa de un cohete, es la relación existente entre la masa de lanzamiento y la masa resultante al final de la misión, es decir reduciendo la masa del propelente y del aislante que son consumidos en el proceso de combustión.

De la ecuación 1 se obtiene dicha relación de masa.

$$RM = \frac{1384{,}728gr}{1778{,}248\text{gr}} = 0{,}778$$

[59] National aeronautics and space administration (NASA), (página consultada el 4 de marzo del 2010), Beginner's Guide to Rockets, http://www.grc.nasa.gov/WWW/K-12/rocket/rktstab.html.
[60] Recuenco. Op.,Cit., p.17.

A partir de lo cual se concluye que la masa de la estructura (masa seca), es el 77,88% de la masa total del vehículo; a su vez se deduce que la masa del propelente y aislante consumidos durante la combustión es del 22,12% de la masa total del cohete.

2.5. Sistema de Propulsión

Como se establece en la sección 2.3.2.4. el material de la Cámara de Combustión, es Aluminio 7075.

La presión máxima admisible en la cámara de combustión, depende del límite elástico o de fluencia del material y de su espesor.[61]

El valor del límite elástico del aluminio 7075 es de 105Mpa.[62]

A través de la ecuación 24, se obtiene el valor de la presión máxima admisible en el la Cámara de Combustión.

Ecuación 24.PresiónMáxima Admisible en la Cámara de Combustión.

$$P_{C-Ad} = \frac{\sigma({r_{Ce}}^2 - {r_{Ci}}^2)}{{r_{Ci}}^2(1 + \frac{{r_{Ce}}^2}{{r_{Ci}}^2})}$$

Fuente:Gómez, Fabián. Leiva, Huindi. ANÁLISIS DEL RENDIMIENTO DEL PROPELENTE SOLIDO TIPO AMATEUR MEDIANTE BALLISTIC EVALUATION MOTOR (BEM) Y SELECCIÓN DE LA TOBERA MAS ADECUADA PARA SU USO EN EL COHETE SONDA LIBERTADOR I. p.39. Bogotá D.C.2015

Donde P_{C-Ad} es la presión máxima admisible en la cámara de combustión, σ esel límite elástico del material, r_{Ce} el radio externo de la Cámara de Combustión y r_{Ci} es el radio interno delaCámara de Combustión.

De esta manera, reemplazando en la ecuación 24, se obtiene el valor de la presión máxima a la que puede operar la cámara de combustión.

$$P_{C-Ad} = \frac{105MPa((0{,}038m)^2 - (0{,}035m)^2)}{(0{,}038m)^2(1 + \frac{(0{,}038m)^2}{(0{,}035m)^2})} = 8{,}62MPa$$

[61] Gomez. Leiva. Op.,Cit., p.39.
[62] ALACER MAS. PROPIEDADES MECANICAS TIPICAS ALUMINIO 7075. [En línea].

Para hallar el valor de la presión de trabajo de la Cámara de Combustión (P_{C-T}), es necesario calcular los valores de Velocidad de descarga característica (C^*) y de *Klemmung (Kn).*

El valor de la velocidad de descarga característica, se obtiene con la ecuación 25.

Ecuación 25. Velocidad de Descarga Característica.

$$C^* = \sqrt{\frac{R'(T_c)}{M(K)}\left(\frac{K+1}{2}\right)^{\frac{K+1}{K-1}}}$$

Fuente: Richard Nakka´s Experimental Rocketry Web Site. Rocket Motor Design Charts -Chamber Pressure [en linea].

Donde R' es la constante universal de los gases (8314 J/mol-K), T_c la temperatura ideal de combustión (1710 K), M el peso molecular efectivo de los productos (42,39 Kg/mol) y K la relación de calor especifico del propelente (1,043). Estos valores corresponden a propelente de Nitrato de potasio con dextrosa, mezcla 65%-35%, respectivamente. (Ver Anexo A).

Reemplazando en la ecuación 25:

$$C^* = \sqrt{\frac{\left(8314\,\frac{J}{mol\,K}\right)(1710\,K)}{\left(42{,}39\,\frac{Kg}{Mol}\right)(1{,}043)}\left(\frac{1{,}043+1}{2}\right)^{\frac{1{,}043+1}{1{,}043-1}}} = 939{,}93\,\frac{m}{s}$$

El Kn es por definición la relación entre el área de combustión (área de quemado de la cámara) y el área de la garganta de la tobera.[63]

El valor de Kn, está dado por la ecuación 26.

Ecuación 26. *Klemmung* del Motor Cohete.

$$Kn = a' + (b(P_{C-Ad}))$$

Fuente: Juan Parczewski´s Amateur Experimental Rocketry WEB SiteDiseño de Motores Cohete en 10 Pasos [en línea].

[63] Juan Parczewski´s Amateur Experimental Rocketry WEB Site. Diseño de Motores Cohete en 10 Pasos. [En línea].

Donde Kn es el valor del Klemmung, P_{C-Ad} la presión máxima admisible en la cámara de combustión, a' y b son coeficientes que dependen de la presión máxima de admisible (8,52 y 44,68/MPa, respectivamente. (Ver Anexo B).

Reemplazando en la ecuación 26, se obtiene el valor del *Klemmung*.

$$Kn = 8{,}52 + \left(\frac{44{,}68}{MPa}(8{,}62MPa)\right) = 393{,}66$$

El área de quemado de la cámara de combustión (A_Q), esta dado por la ecuación 27.

Ecuación 27. Área de Quemado de la Cámara de Combustión.

$$A_Q = 2\pi(l_{Cas})(r_{Casi})$$

Fuente: OnlineMSchool.Área de Cilindro [en línea].

Donde l_{Cas} es la longitud de la Cámara de Combustión y r_{Casi} el radio interior de la Cámara de Combustión.

Así, el valor del área de quemado es:

$$A_Q = 2\pi(25cm)(1{,}75cm) = 274{,}889cm^2$$

Como se determinó anteriormente, Kn es la relación entre el área de quemado de la cámara de combustión (A_Q) y el área de la garganta de la tobera (A_G). Por lo anterior, la ecuación 28 permite obtener el valor del área de la garganta.

Ecuación 28. Área de la Garganta de la Tobera.

$$A_G = \frac{A_Q}{Kn}$$

Fuente: Richard Nakka´s Experimental Rocketry Web Site. Burn Rate Determination from a Pressure-time Trace [en linea].

De acuerdo a lo anterior, el área de la garganta de la tobera es:

$$A_G = \frac{274{,}889cm^2}{393{,}66} = 0{,}698cm^2$$

El radio de la garganta, está dado por la ecuación 29.

Ecuación 29. Radio de la Garganta de la tobera.

$$\boldsymbol{r_G = \sqrt{\frac{A_G}{\pi}}}$$

Fuente: Calcular Área. Calcular Área del Círculo [en línea].

De esta forma, el radio de la garganta de la tobera es:

$$\boldsymbol{r_G = \sqrt{\frac{0{,}698cm^2}{\pi}}} = 0{,}471cm$$

El diámetro de un circulo, por definición es dos veces la longitud del radio del mismo.[64] Por lo tanto el diámetro de la garganta de la tobera, es:

$$\emptyset_G = 2(0{,}471cm) = 0{,}942cm$$

La presión de trabajo de la Cámara de Combustión está dada por la ecuación 30.

Ecuación 30. Presión de Trabajo en la Cámara de Combustión.

$$P_{C-T} = \left(Kn\left(\frac{a}{\alpha}\right)(\rho_P)(C^*)\right)^{\frac{1}{1-n}}$$

Fuente: Richard Nakka´s Experimental Rocketry Web Site. Motor Design Charts - Chamber Pressure [en linea].

Donde P_{C-T} es la presión de trabajo en la cámara de combustión, Kn el klemmung del motor cohete, ρ_P la densidad del propelente, C^* la vlocidad de descarga característica, α es el factor de conversión de Pa a MPa (1'000.000), a el

[64] Editorial Norma. Enciclopedia Tematica Ilustrada. La Aventura del Conocimiento.

coeficiente de combustión bajo presión (4,78mm/s) y n el exponente de combustión bajo presión (0,442). Los valores de a y n, son dependientes de la presión máxima admisible en la cámara. (Ver Anexo C).

Reemplazando en la ecuación 30, se obtiene el valor de la presión de trabajo en la Cámara de Combustión.

$$P_{C-T} = \left((393{,}66) \left(\frac{\left(\frac{4{,}78mm}{s}\right)\frac{1m}{1000mm}}{1'000.000} \right) (1857Kg/m^3)(939{,}93m/s) \right)^{\frac{1}{1-0{,}0442}}$$

$$P_{C-T} = 8{,}425MPa$$

El impulso especifico (I_{SP}), que el propelente es capaz de producir, es la clave del potencial de rendimiento. Este impulso específico es el número de segundos que un Kg de propelente, proporciona 1 Kg de empuje.[65]

La ecuación 31, permite obtener el valor de dicho impulso específico.

Ecuación 31. Impulso Especifico del Propelente.

$$I_{SP} = \frac{1}{g}\sqrt{2T_C\left(\frac{R'}{M}\right)\left(\frac{K}{K-1}\right)\left[1-\left(\frac{P_e}{P_{C-T}}\right)^{\frac{K-1}{K}}\right]}$$

Fuente: Nakka Richard. Teoría Sobre Motores Cohete de Propelente Solido. Traducido por Garofalo. Sebastián [en línea]

Donde g es el valor de la aceleración de la gravedad (9,81m/s) yP_e es la presión de salida o presión exterior entorno al motor (0,0738 Mpa en Bogotá[66]).

Por lo anterior, el impulso específico del propelente es:

[65] Nakka. Richard. Teoria Sobre Motores Cohete de Propelente Solido. Traducido por Garofalo. Sebastian. [En línea].
[66] Anderson. John. INTRODUCTION TO FLIGHT. McGraw-Hill. seriesTercera Edicion. 1989.

$$I_{SP} = \frac{1}{9{,}81\frac{m}{s}}\sqrt{2(1710\ K)\left(\frac{8314\frac{J}{mol\ K}}{\frac{42{,}39Kg}{mol}}\right)\left(\frac{1{,}043}{1{,}043-1}\right)\left[1-\left(\frac{0{,}0738MPa}{8{,}425MPa}\right)^{\frac{1{,}043-1}{1{,}043}}\right]}$$

$$I_{SP} = 173{,}15s$$

El impulso total (I_{Tot}), es la energía total que un motor produce durante el tiempo total de combustión del propelente.[67] El valor de este, está dado por la ecuación 32.

Ecuación 32. Impulso Total del Motor Cohete

$$I_{Tot} = I_{SP}(W_P)$$

Fuente:Nakka Richard. Teoría Sobre Motores Cohete de Propelente Solido. [en línea]

Donde W_P, es el peso del propelente.

Remplazando los valores en la ecuación 32, se obtiene el impulso total.

$$I_{Tot} = (173{,}15s)((0{,}364174\text{Kg})\left(9{,}81\frac{\text{m}}{s^2}\right) = 618{,}604\ Ns$$

El valor del impulso total, indica que tipo de motor cohete es el motor en proceso de diseño. Al tener el valor de $618{,}604\ Ns$, indica que es un motor tipo I (320 Ns-640 Ns).[68]

La relación de presión, es la relación que existe entre la Presión exterior (P_e) y la presión de trabajo de la cámara de combustión (P_{C-T}).[69] Su expresión está dada por la ecuación 33.

[67]Riveros. Felipe, Rodriguez. Luis. DISEÑO Y CONSTRUCCIÓN DE UN COHETE AFICIONADO CONTROLADO MEDIANTE EL ACCIONAMIENTO DE UNA TOBERA DE EMPUJE VECTORIAL. Bogota. 2010. p. iiV.
[68] Richard Nakka's Experimental Rocketry Web Site. Rocket Motor Letter Code. [En linea].
[69] Correal. M, Castaño. A, Rincón. O y Aguillón. H. DISEÑO Y CONSTUCCION DE UN COHETE AMATEUR TIPO G. Bogota. 2009. p.42.

Ecuación 33.Relación de Presión

$$RP = \frac{P_e}{P_{C-T}}$$

Fuente: Nakka Richard. Teoría Sobre Motores Cohete de Propelente Solido. [en línea].

De esta forma, reemplazando en la ecuación 33, se obtiene:

$$RP = \frac{0{,}0738MPa}{8{,}425MPa} = 0{,}008759$$

Con el valor de RP obtenido, y al tabular para este valor, en la tabla de coeficiente de empuje para cohetes (ver Anexo D), el valor de Coeficiente de Empuje (C_e), es de 1,658, y el valor de Numero Mach de salida (M_S), es de 3,468.

El empuje se define como la fuerza producida por un motor durante su quemado.[70] Su valor, está dado por la ecuación 34.

Ecuación 34. Empuje del Motor.

$$T_M = (P_{C-T})(A_G)(C_e)$$

Fuente: Nakka Richard. Teoría Sobre Motores Cohete de Propelente Solido. [en línea].

Donde T_M, es el empuje del motor cohete.

Reemplazando en la ecuación 34, se obtiene el valor del empuje:

$$T_M = (8{,}425MPa)\left(1'000.000\frac{Pa}{MPa}\right)(0{,}698cm^2)\left(\frac{1m^2}{(100cm)^2}\right)(1{,}658)$$

$$T_M = 975{,}011N$$

[70] Nakka. Richard. Teoria Sobre Motores Cohete de Propelente Solido. Traducido por Garofalo. Sebastian. [En línea].

El flujo másico($\dot{m}$), es la cantidad de masa de propelente que se expulsa de la cámara de combustión a través de la tobera, en el transcurso del tiempo de combustión. La ecuación 35, permite obtener este valor.

Ecuación 35. Flujo Másico.

$$\dot{m} = \frac{T_M}{I_{SP}(g)}$$

Fuente: Nakka Richard. Teoría Sobre Motores Cohete de Propelente Solido. [en línea].

Reemplazando los valores en la ecuación 35, se obtiene el flujo másico.

$$\dot{m} = \frac{975{,}011N}{(173{,}15s)\left(9{,}81\frac{m}{s}\right)} = 0{,}574\frac{Kg}{s}$$

A través del flujo másico, puede determinarse el tiempo de quemado del propelente, como se muestra en la ecuación 36.

Ecuación 36. Tiempo de Quemado del Propelente.

$$t = \frac{m_P}{\dot{m}}$$

Fuente: Brito. Jean Frank. DISEÑO Y EVALUACION CONCEPTUAL DE UN MOTOR COHETE DE COMBUSTIBLE SOLIDO. Sartenejas. 2011.

Así, el tiempo de quemado es:

$$t = \frac{0{,}36417Kg}{0{,}574\frac{Kg}{s}} = 0{,}6344s.$$

Para determinar el área de salida, es necesario utilizar la ecuación de flujo isentrópico la cual relaciona el Mach y la relación de calor especifico con la relación de áreas.

Ecuación 37. Relación de Área de Salida y Área de Garganta de Tobera.

$$\frac{A_s}{A_G} = \left(\frac{K+1}{2}\right)^{-\frac{K+1}{2(K-1)}} \left(\frac{(1+\frac{K-1}{2}(M_S)^2)^{\frac{K+1}{2(K-1)}}}{M_S}\right)$$

Fuente: NASA. Isentropic Flow. [en línea]

Al despejar el área de salida y reemplazar los valores en la ecuación 37, se obtiene el valor del área transversal de salida en la tobera.

$$A_s = (0{,}698cm^2)\left(\frac{1{,}043+1}{2}\right)^{-\frac{1{,}043+1}{2(1{,}043-1)}} \left(\frac{(1+\frac{1{,}043-1}{2}(3{,}468)^2)^{\frac{1{,}043+1}{2(1{,}043-1)}}}{3{,}468}\right)$$

$$A_s = 28{,}632cm^2$$

Con base en la ecuación 29, se obtiene el valor del radio de salida de la tobera.

$$r_S = \sqrt{\frac{28{,}632cm^2}{\pi}} = 3{,}018cm$$

$$\emptyset_S = 2(3{,}018cm) = 6{,}036cm$$

Para las secciones convergente y divergente de la tobera, se debe tener en cuenta que el uso de ángulos muy pronunciados puede inducir flujo turbulento, sin embargo, también se debe tener en cuenta que el uso de ángulos muy suaves alarga las dimensiones de la tobera, aumentando así el peso de la estructura, reduciendo por esto la eficiencia del motor. El ángulo de la sección convergente suele estar entre 30 y 60 grados, teniendo la mayoría de diseños, valores cercanos a los 45°.

Del mismo modo la sección divergente se encuentra entre 10 y 15 grados, con la mayoría de diseños cerca a los 12°.[71]

Por lo anterior se decide utilizar para la sección convergente, ángulo de 45° y para la sección divergente de 12°.

[71] PEREZ. E, BERMEO. N Y FÚQUEN. D. DISEÑO Y CONSTRUCCION DE UN MOTOR COHETE QUE PRODUZCA 800N DE EMPUJE. Bogota. 2010.

En la tabla 7, se muestran las características de la tobera.

Tabla 7. Características de la Tobera del ARISTARCO I.

TOBERA ARISTARCO I		
Característica	**Valor**	**Unidad**
Diámetro de Entrada	3,5	cm
Diámetro de Garganta	0,942	cm
Diámetro de Salida	6,036	cm
Angulo Sección Convergente	45	°
Angulo Sección Divergente	12	°

Fuente: Autores.

2.6. Apogeo

La velocidad del sonido está dada por la ecuación 38.

Ecuación 38. Velocidad de Propagación del Sonido.

$$a_S = \sqrt{\gamma R T}$$

Fuente: Anderson. John. INTRODUCTION TO FLIGHT. McGraw-Hill. SeriesTercera Edición. 1989.

Donde γ, es la relación de calores específicos a presión y volumen constantes (1,4 para el aire), R la constante del gas (286,9 J/Kg K para el aire) y T es la temperatura del aire (271,27 para la altura de Bogotá).[72]

Reemplazando en la ecuación 38, se obtiene el valor de la velocidad de propagación del sonido a la altura de Bogotá.

$$a_S = \sqrt{(1{,}4)\left(286{,}9\,\frac{J}{Kg\ K}\right)(271{,}27K)} = 330{,}08\,\frac{m}{s}$$

El numero Mach está dado por la ecuación 39.

Ecuación 39. Numero Mach.

[72]Anderson. John. INTRODUCTION TO FLIGHT. McGraw-Hill. Series Tercera Edicion. 1989.

$$M_{ach} = \frac{V}{a_S}$$

Fuente: Anderson. John. INTRODUCTION TO FLIGHT. McGraw-Hill. SeriesTercera Edición. 1989.

Donde V, es el valor de una velocidad que se quiera relacionar con a_S, que es la velocidad de propagación del sonido en un medio determinado.

Despejando la velocidad de la ecuación 39 y reemplazando en esta el valor del número Mach de salida de tobera, se puede determinar la velocidad de salida de la tobera del motor.

$$V_S = \left(330{,}08\frac{m}{s}\right)(3{,}468) = 1144{,}71\frac{m}{s}$$

La velocidad máxima del cohete, está dada por la ecuación 40.

Ecuación 40. Velocidad Máxima del Cohete.

$$V_{Max} = -g(t) + (V_S)\left(\ln\left(\frac{m_i}{m_i - \dot{m}(t)}\right)\right)$$

Fuente: Sistema de Masa Variable. Movimiento Vertical de un Cohete. [en línea]

Reemplazando los valores correspondientes en la ecuación 40, se obtiene la velocidad máxima a la que vuela el *ARISTARCO I.*

$$V_{Max} = \left(-9{,}81\frac{m}{s^2}\right)(0{,}6344s) + \left(1144{,}71\frac{m}{s}\right)\left(\ln\left(\frac{1{,}778248Kg}{1{,}778248Kg - 0{,}364Kg}\right)\right)$$

$$V_{Max} = 256{,}09\frac{m}{s}$$

De la ecuación 39, se obtiene el numero Mach máximo al que vuela el cohete.

$$M_{ach} = \frac{256{,}09\frac{m}{s}}{330{,}08\frac{m}{s}} = 0{,}775$$

La altura máxima que alcanza el cohete en su vuelo, se determina en dos secciones; una calculando la altura que alcanza el cohete mientras dura la combustión del propelente y segunda, calculando la altura que alcanza el cohete después de terminar la combustión.

La ecuación 41, permite obtener el valor de la altura que alcanza el cohete hasta que termina el proceso de combustión.

Ecuación 41. Altura que Alcanza el Cohete Durante la Combustión.

$$h_{comb} = -\frac{1}{2}(g)(t) + (V_S)(t)(\ln(m_i)) + \frac{V_S}{\dot{m}}\left(m_i - (\dot{m})(t)\right)\ln\left(m_i - (\dot{m})(t)\right) + (\dot{m})(t) - (m_i)(\ln(m_i))$$

Fuente: Sistema de Masa Variable. Movimiento Vertical de un Cohete. [en línea].

Reemplazando:

$$h_{comb} = -\frac{1}{2}\left(9{,}81\frac{m}{s^2}\right)(0{,}6344) + \left(1144{,}71\frac{m}{s}\right)(0{,}6344s)(\ln(1{,}778248kg)) + \frac{1144{,}71\frac{m}{s}}{0{,}574\frac{Kg}{s}}\left(1{,}778248Kg - \left(0{,}574\frac{Kg}{s}\right)(0{,}6344s)\right)\ln(1{,}778248Kg - \left(0{,}574\frac{Kg}{s}\right)(0{,}6344s) + \left(0{,}574\frac{Kg}{s}\right)(0{,}6344s) - (1{,}778248Kg)(\ln(1{,}778248Kg)))$$

$$h_{comb} = 78{,}03m$$

De la ecuación 42, se obtiene el tiempo total de ascenso del vehículo.

Ecuación 42. Tiempo de ascenso del Cohete.

$$t_{Tot} = \frac{V_{Max}}{g} + t$$

Fuente: Sistema de Masa Variable. Movimiento Vertical de un Cohete. [en línea].

Reemplazando se tiene:

$$t_{Tot} = \frac{256{,}09\frac{m}{s}}{9{,}81\frac{m}{s^2}} + 0{,}6344s = 26{,}739s$$

La altura máxima que alcanza el cohete, se obtiene mediante la ecuación 43.

Ecuación 43. Altura Máxima del Cohete en Vuelo.

$$h_{Max} = h_{Comb} + (V_{Max})(t_{Tot} - t) - \frac{1}{2}(g)(t_{Tot} - t)^2$$

Fuente: Sistema de Masa Variable. Movimiento Vertical de un Cohete. [en línea].

Reemplazando en la ecuación 43, se tiene:

$$h_{Max} = 78{,}03m + \left(256{,}09\frac{m}{s}\right)(26{,}739s - 0{,}6344s) - \frac{1}{2}(9{,}81\ m/s^2)(26{,}739s - 0{,}6344s)^2$$

$$h_{Max} = 3421{,}569m$$

De esta manera se establece que el apogeo del *ARISTARCO I,* se encuentra alrededor de los 3,4Km.

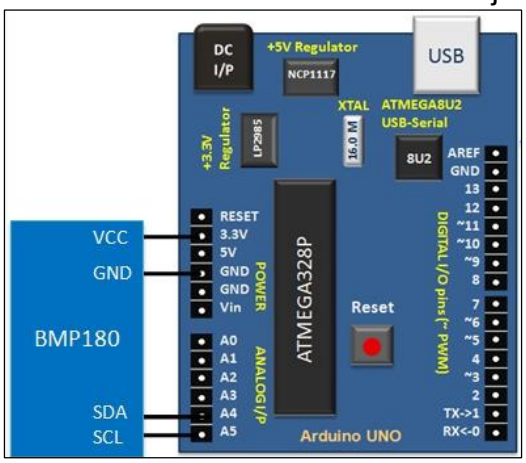
2.7. Carga Paga y Sistema de Recuperación.

2.7.1. Programación

Como se especificó en la sección 1.8. la plataforma utilizada para desarrollar el sistema de adquisición de datos y el sistema de recuperación es ARDUINO, con la tarjeta ARDUINO UNO. A través de esta plataforma, se desarrolla dos códigos que reúne los sensores descritos en las secciones 1.6. y 1.9. y obtiene los datos pertinentes para el desarrollo de la misión del ARISTARCO I. Un código pertenece al sistema de carga paga e instrumentación, el cual permite obtener y almacenar los datos de las condiciones atmosféricas altitud y tiempo. El segundo código es destinado al sistema de recuperación el cual permite obtener la información de inclinación del cohete, y así cuando detecte que se realizando el giro para precipitarse a tierra, es decir, cuando el vehículo tiene una inclinación mayor a 75° con respecto al eje vertical, se activaran los motores encargados de expulsar el paracaídas. Adicional a esto, por si existe alguna condición en la que el cohete inicie el descenso en una posición, en la cual el vehículo no supere el ángulo establecido, el sistema de recuperación se activara 2,5 minutos después de energizado el sistema en tierra, durante las operaciones de lanzamiento; dando así, dos minutos para preparar el lanzamiento y 30 segundos de duración de ascenso del vehículo. (Anexo X, Anexo Y).

Mediante el sensor BMP-180, se obtienen los valores de presión atmosférica, temperatura, altitud sobre el nivel del mar y altitud sobre un punto de referencia, el cual se determina como el lugar de lanzamiento. La implementación del sensor en la tarjeta ARDUINO, utiliza las librerías BMP180, I2C y WIRE (Anexos S, T, U); su conexión a la tarjeta se realiza como se evidencia en la imagen 24.

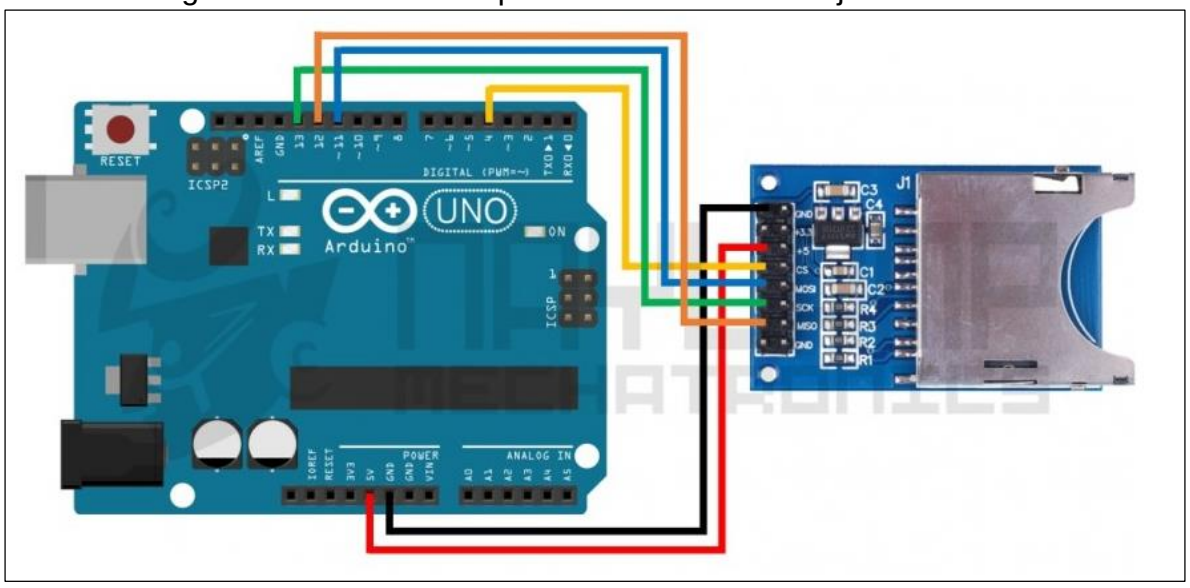
Imagen 24. Conexión Sensor BMP-180 con Tarjeta ARDUINO.



Fuente: Embedded Lab. USING BMP180 FOR TEMPERATURE, PRESSURE AND ALTITUDE MEASUREMENTS. [En línea].

Para lograr almacenar los datos obtenidos por el sensor BMP 180, se utiliza una tarjeta MicroSD, la cual se incorpora a la tarjeta ARDUINO mediante un adaptador de tarjeta MicroSD, cuya conexión está dada en la imagen 25. La librería utilizada para realizar el almacenamiento de datos es la librería SD (ANEXO V).

Imagen 25. Conexión Adaptador MicroSD con Tarjeta ARDUINO.

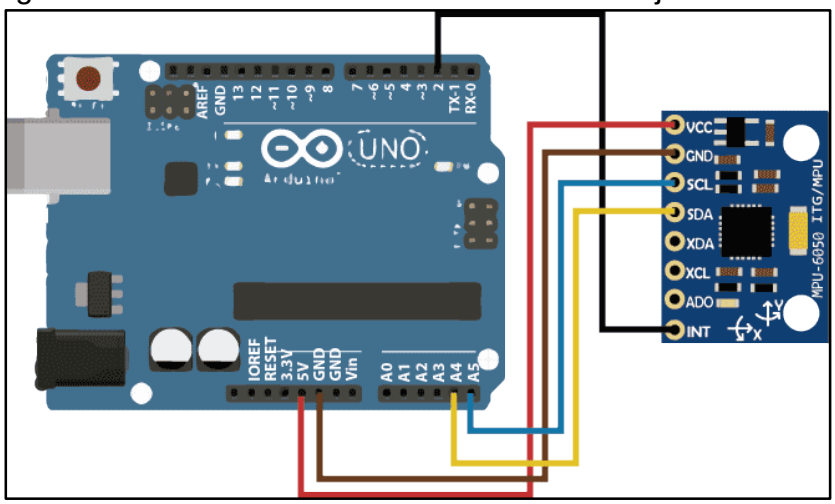


Fuente: NAYLAMPS MECHATRONICS. Tutorial Arduino y memoria SD y micro SD. [En línea].

A través del sensor MPU 6050, se obtiene la inclinación que tiene el cohete con respecto a los ángulos X*, Y* y *Z.* Una vez el cohete alcanza su apogeo y empieza el descenso, la orientación del vehículo cambia. Por esto y a su vez, con fin de evitar un incidente por desviación en la trayectoria del vehículo durante el vuelo, una vez dicho cambio de orientación alcanza una inclinación que supera los 15° con respecto al eje *X* y *Y*, ya que el sensor está ubicado verticalmente en el cohete, se activa el sistema de recuperación, expulsando así el paracaídas mediante un mecanismo de piñón-cremallera activado por dos motores alimentados por 5 voltios.

La implementación del sensor en la tarjeta ARDUINO, utiliza las librerías MPU 6050, I2C y WIRE (Anexos W, T y U); su conexión a la tarjeta se realiza como se evidencia en la imagen 26.

Imagen 26. Conexión Sensor MPU-6050 con Tarjeta ARDUINO.



Fuente: DIYhacking. ARDUINO MPU 6050. [En línea].

2.7.2. Paracaídas

Como se describió en la sección 1.6. el paracaídas que se implementara en el cohete *ARISTARCO I* es el paracaídas disponible en el semillero de investigación AERODES&I, el cual posee un diámetro de 1,5 metros. La velocidad adecuada de descenso de un cohete es de aproximadamente 7 m/s

La velocidad de descenso estable producida por un paracaídas está dada por la ecuación 44.

Ecuación 44. Velocidad de descenso del Cohete

$$V_{Des} = \sqrt{\frac{2 * W_t}{S_p * C_d * \rho_{aire}}}$$

Fuente: Richard Nakka´s Experimental Rocketry Web Site. Parachute Design and Construction. [en línea]

Donde (W_t) es el peso total del cohete, (S_p) es la superficie del paracaídas, (C_d) es el coeficiente de Drag del paracaídas (cuyo valor es de 0,75[73]) y (ρ_{aire}) es la densidad del aire

Reemplazando los valores en la ecuación 44:

$$V_{Des} = \sqrt{\frac{2 * \left(1{,}778248\ kg * 9{,}81 \frac{m}{s^2}\right)}{(\pi * (0{,}75m)^2) * (0{,}75) * \left(1{,}225 \frac{kg}{m^3}\right)}} = 4{,}014\ \frac{m}{s}$$

La velocidad obtenida, se encuentra por debajo de la velocidad adecuada de descenso anteriormente mencionada, por lo tanto, es aceptable para implementar el paracaídas mencionado en el cohete ARISTARCO I.

[73] Parks College Parachute Research Group, Calculating the descent rate of a round parachute [en línea]

3. DISEÑO DETALLADO

El diseño detallado como proceso de ingeniería básico, transforma alternativas de conceptos, especificaciones de diseño y requisitos técnicos en definiciones de diseño finales e interdisciplinares; estos diseños se ajustan más y se elabora toda la documentación que les acompaña y que necesita para la posterior fabricación con el fin de entregar un producto completo y totalmente definido[74].

En concordancia a los resultados obtenidos en la fase de diseño preliminar, permite el desarrollo de la etapa final del ARISTARCO I, el Diseño Detallado se desarrolla a través de modelos planteados en CAD de cada una de las piezas las cuales conforman el cohete sonda, al igual que su proceso de manufactura para dichos componentes característicos del ARISTARCO I, y el ensamble de cada uno de los componentes en el cohete.

3.1. Estructura

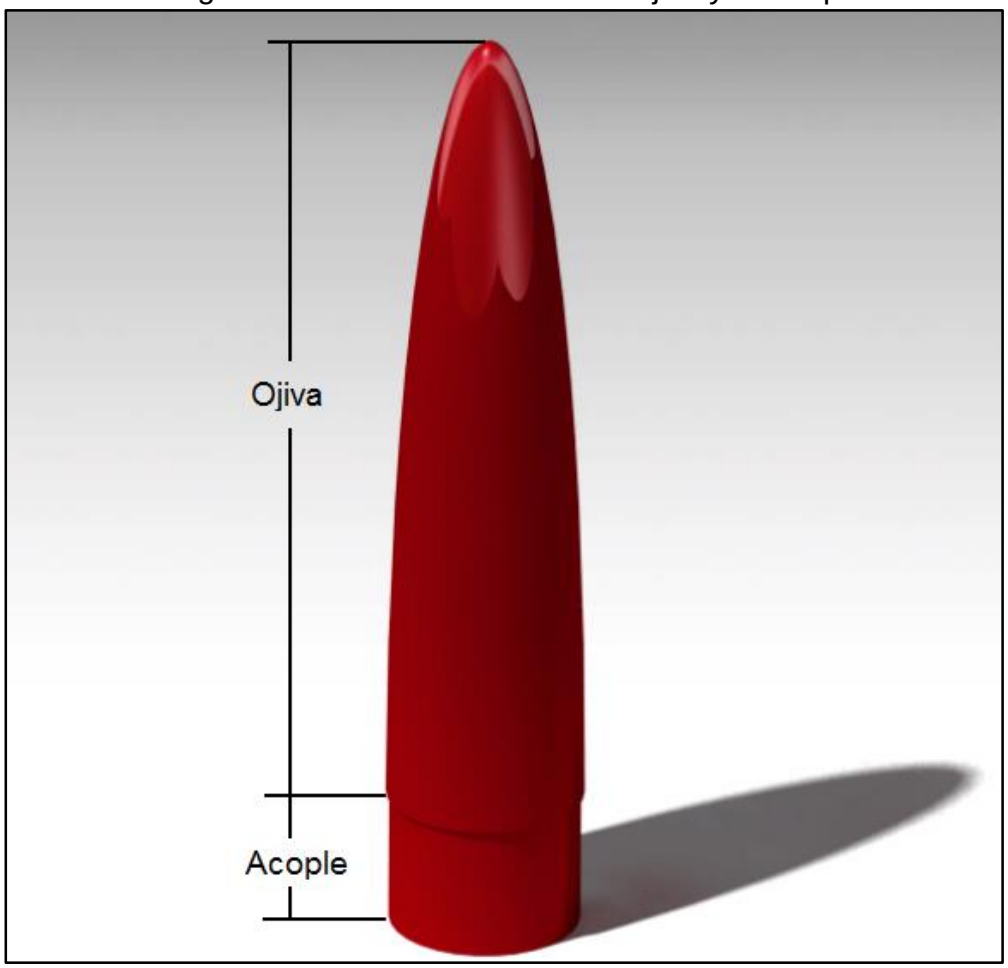
3.1.1. Ojiva

De acuerdo a los datos obtenidos en la sección 2.2.1., con una longitud de Ojiva (l_o) de 30cm se realiza el diseño en CAD para obtener la curva característica de la Ojiva, con un diámetro externo de su base de 7.2cm, y un espesor de pared de 0.4 cm (Ver Anexo E).

En la siguiente imagen se puede observar el diseño finalizado de la ojiva, en este se encuentra en la parte inferior una sección cilíndrica con un menor diámetro que el diámetro externo de la base de la ojiva, esta sección es empleada para realizar el acople con el fuselaje del cohete sonda ARISTARCO I.

[74] PTC. Hoja Temática:Diseño Detallado [en línea]

Imagen 27. Vista Isométrica de la Ojiva y el Acople

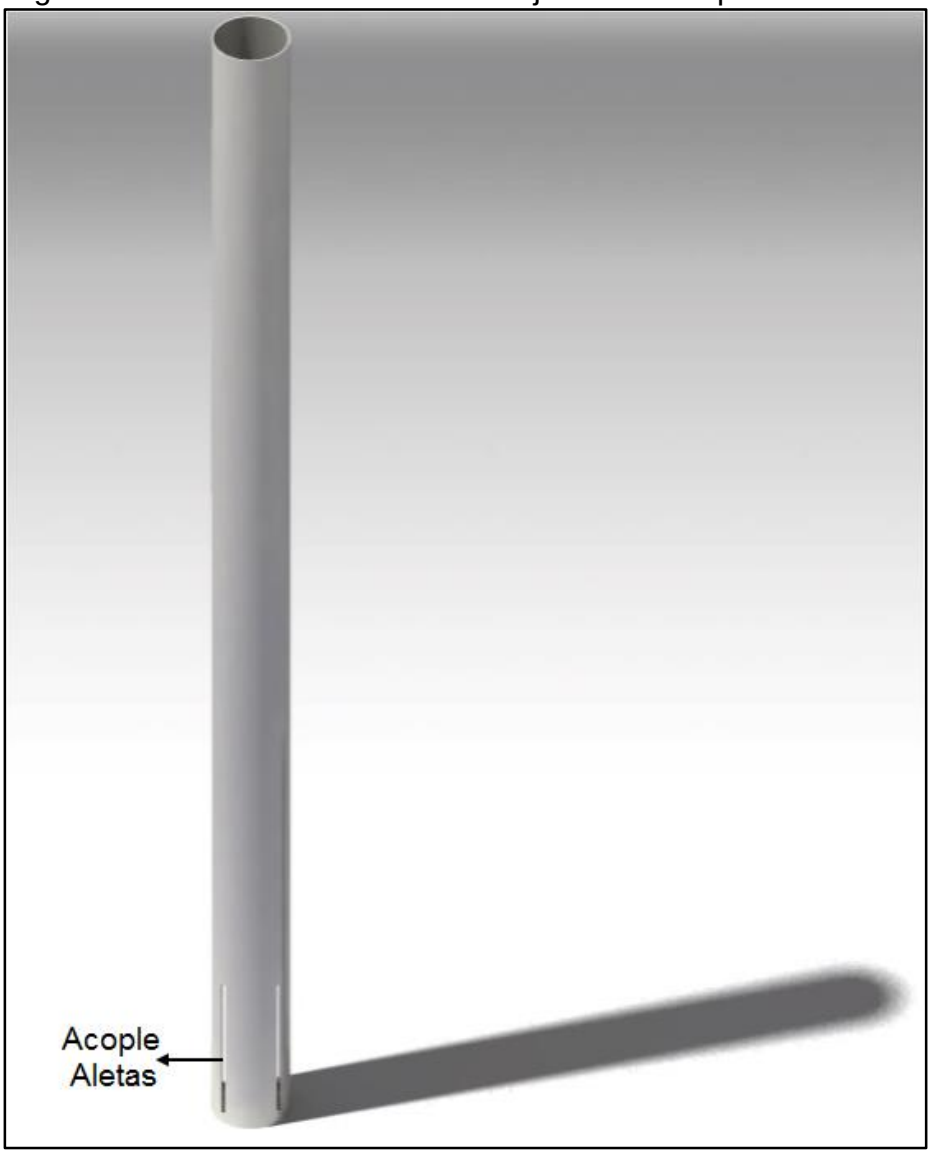


Fuente: Autores

3.1.2. Fuselaje

Teniendo en cuenta la dimensión determinada para el fuselaje de 120cm, se realiza el diseño final: hay que tener en cuenta que en el Fuselaje (F) en su sección externa va ensamblado el componente de las aletas que en su totalidad son 4, por lo tanto, se observa aberturas en la parte inferior del fuselaje de aproximadamente 0.4cm (Ver Anexo F) para ingresar una sección adicional del diseño de las aletas para tener una sujeción interna.

Imagen 28 .Vista Isométrica del Fuselaje con el Acople de las Aletas



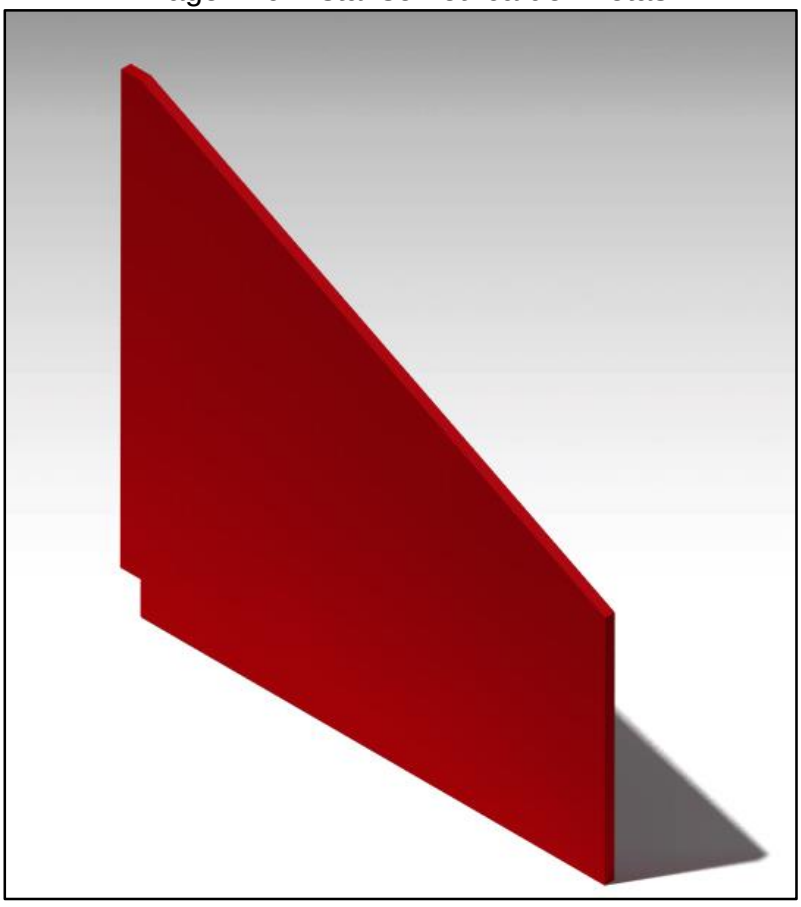
Fuente: Autores

El tubo de fibra fenólica tiene un espesor de 1.325cm y se le aplicaron dos capas de pinturapara acabados finales, adicionalmente se le realizaron 4 cortes longitudinales para realizar el acople de las aletas (Sección 3.2.3.), posee una longitud de 14.4cm y espesor de 0.4cm (Ver Anexo F).

3.1.3. AletasEstabilizadoras

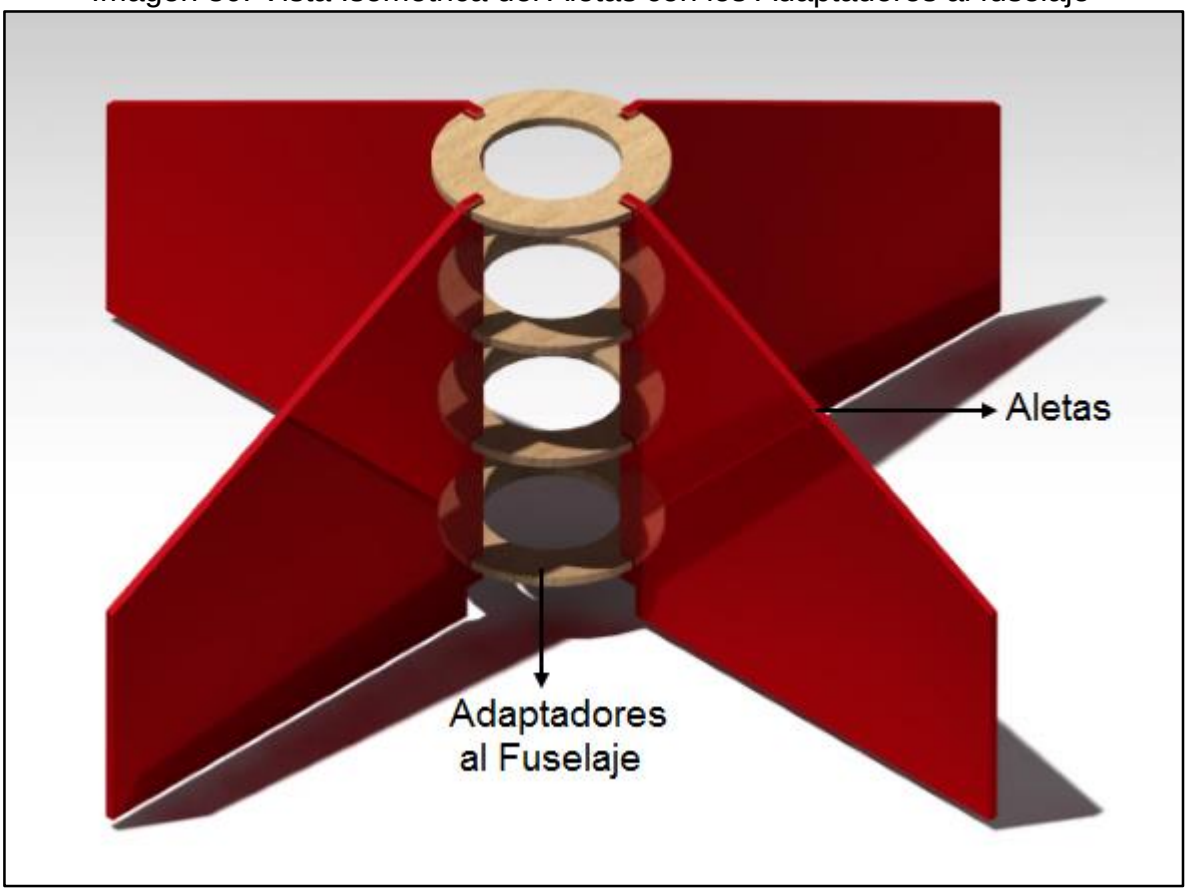
Las aletas del ARISTARCO I son dimensionadas de acuerdo a los parámetros resultantes de la sección 2.2.3., de igual manera se le ha agregado un área a la cuerda de raíz (Ver Anexo G) de las aletas con el fin de cumplir la función de adaptación al fuselaje, como se puede observar en la imagen 29.

Imagen 29.Vista Isométrica del Aletas



Fuente: Autores

Imagen 30. Vista Isométrica del Aletas con los Adaptadores al fuselaje



Fuente: Autores

Como se puede detallar en la anterior imagen (Imagen 30), se encuentran los adaptadores de las aletas al fuselaje, las cuales también cumplen dos funciones adicionales, y se encuentran directamente conectados con el Fuselaje, y el tubo adaptador de la Cámara de Combustión (ver sección 3.2.5.1.), estos adaptadores cuentan con un espaciamiento para ingresar cada una de las aletas y fijarlas, estos se encuentran cada 90° alrededor de su diámetro exterior (Ver Anexo H), las aletas cuentan con 4 adaptadores al fuselaje los cuales se encuentran ubicados verticalmente cada 4.06 cm aproximadamente uno del otro(Ver Anexos H).

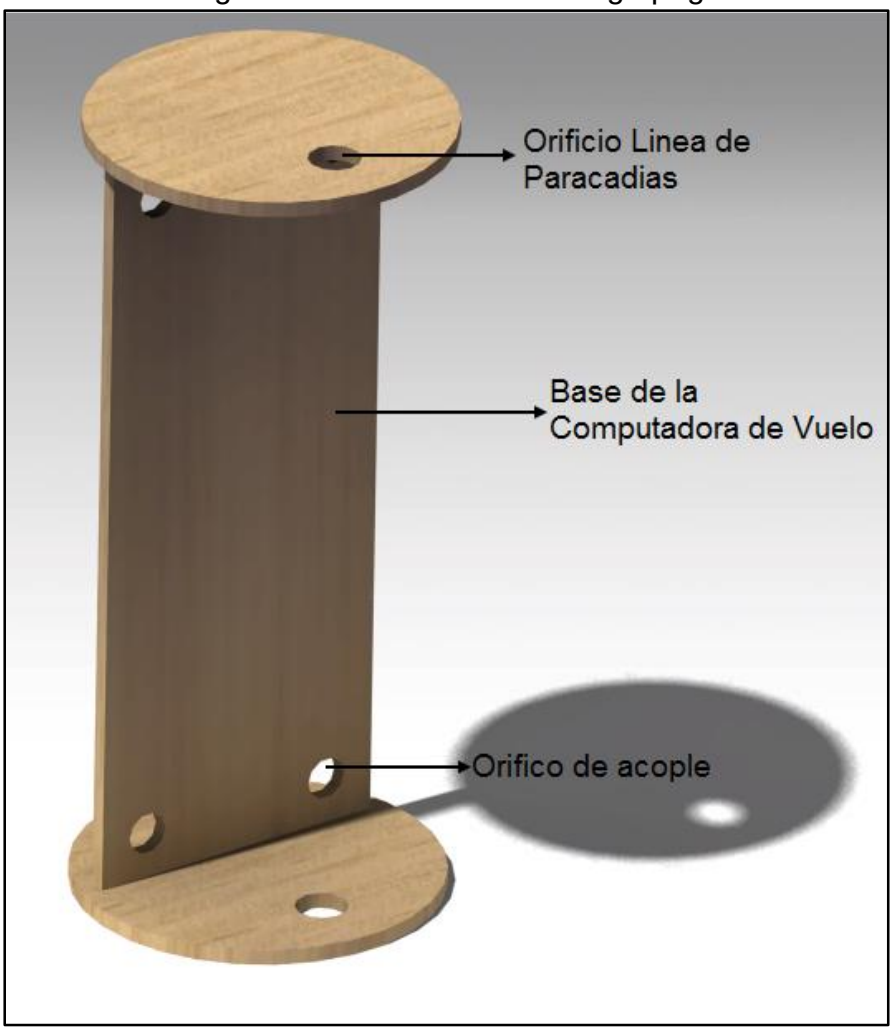
3.1.4. Carga Paga

La carga paga va alojada en una estructura adecuada para su protección y efectivo funcionamiento dentro del cohete ARISTARCO I, como se puede observar a continuación tanto la estructura y su acople con la computadora de vuelo.

3.1.4.1. Estructura Carga Paga

La estructura para el acople de la carga paga se encuentra dimensionada de acuerdo a como se puede observar en la sección 2.3.3.11.2. con dos cubiertas de un espesor de 0.3cm cada una y un diámetro de 6.935 cm, para ser protegida de los otros sistemas, como se puede observar en la siguiente imagen.

Imagen 31. Estructura de la carga paga.



Fuente: Autores

Como se puede observar en la imagen 31, en las cubiertas se encuentra un orificio de 0.5 cm (Ver Anexo I) con la función de permitir el paso de una línea del paracaídas que va sujetada al motor (Ver sección 3.3), adicionalmente cuenta con 4 orificios en la Base, con el fin de poder sujetar la computadora de vuelo a la base de la estructura. Esta estructura se encuentra ubicada de manera longitudinal a través del fuselaje.

3.1.5. Motor

El motor del cohete ARISTARCO I como se ha mencionado anteriormente comprende tres elementos principales para su acople y seguro funcionamiento. Al observar en la sección 2.3.3.11.1.4. se ha logrado dimensionar correctamente tanto la tobera, en la Cámara de Combustión (sección 2.3.3.7.1) y el propelente (P) (sección 2.3.3.7.3) con propiedades características para cumplir efectivamente la misión del ARISTARCO I.

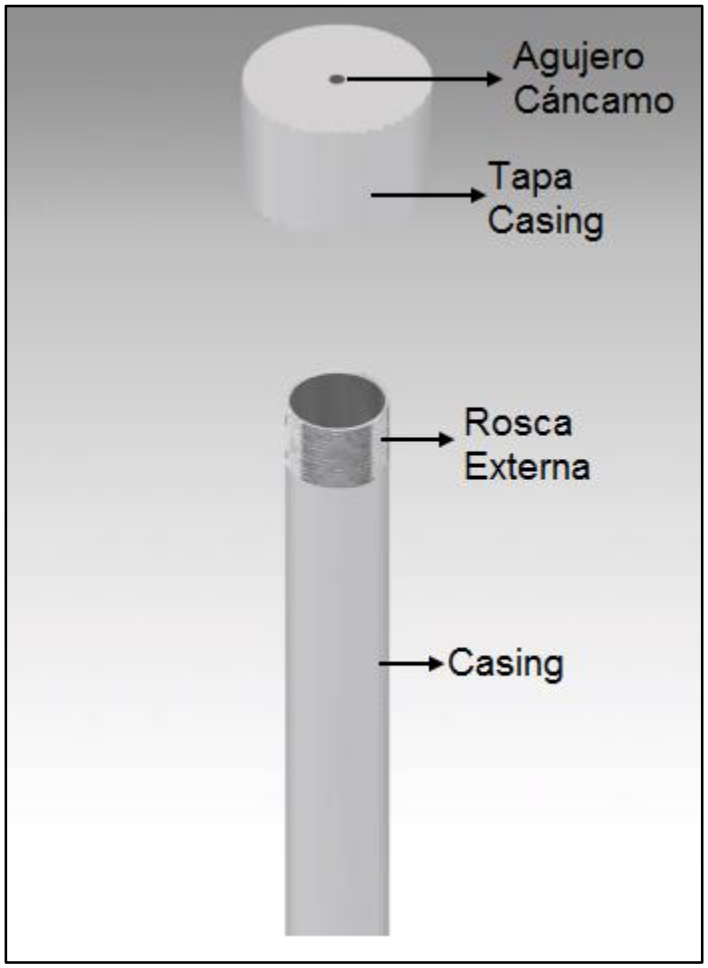
A continuación, se pueden observar diferentes imágenes, detalladamente en las cuales se considera el acople de cada uno de los elementos del Motor al Cohete ARISTARCO I.

3.1.5.1. Cámara de Combustión

En la Cámara de Combustión se encuentra directamente involucrado con diferentes componentes del cohete, como lo son el propelente, tobera, anillos de sujeción, tapa superior, pared de fuego; se puede observar en la Imagen 32se encuentra una tapa en la parte superior del componente, este entra enroscada con la pared externa de la Cámara de Combustión de modo que se disminuye considerablemente pérdidas de presión dentro de la cámara de combustión, ya que adicionalmente contiene un anillo O-Ring en su interior.(Ver Anexo J, Anexo H).

Como se puede observar en la Imagen 32, se encuentra un agujero denominado “agujero cáncamo” hace referencia al agujero donde va atornillado el Cáncamo que fija al motor con una línea del paracaídas, posteriormente la tapa de la Cámara de Combustión, en su interior parte inferior tiene el agujero para entrar enroscado con la rosca externa de la Cámara de Combustión y así asegurar la tapa con la Cámara de Combustión.

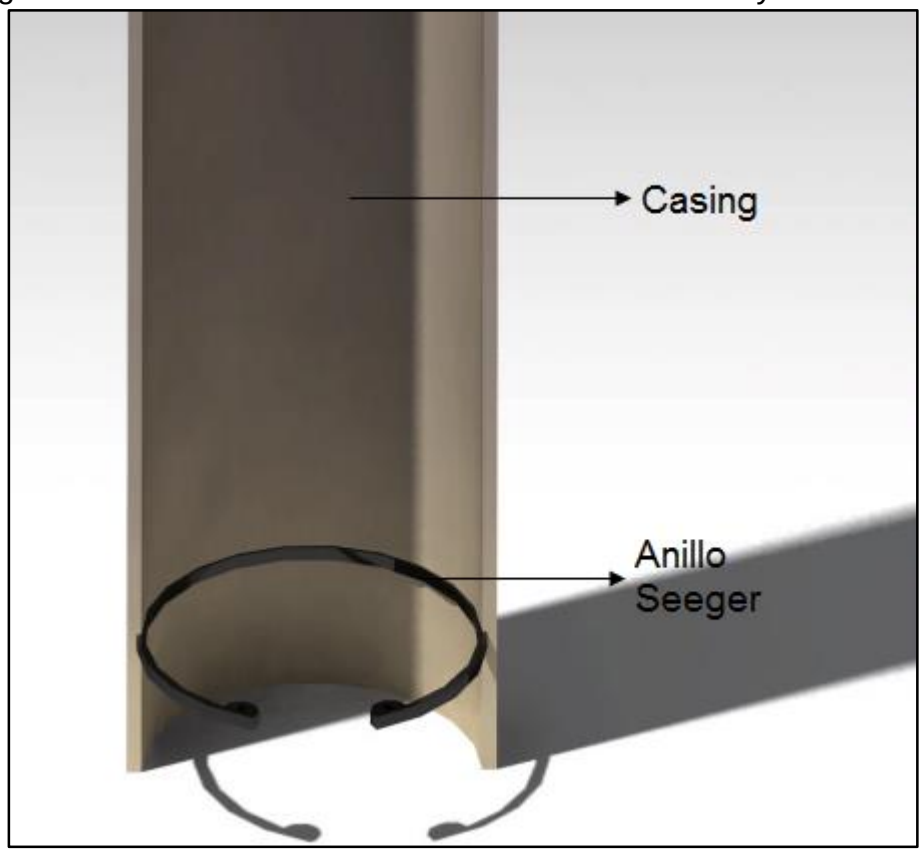
Imagen 32. Vista Isométrica de Cámara de Combustión con su Tapa Superior



Fuente: Autores

En la imagen 33, se puede observar un anillo seeger en la parte inferior de la Cámara de Combustión en su pared interna, la función de dicho anillo es evitar la expulsión de la tobera por las altas presiones, y asegura la sección convergente de la tobera, al igual que permite introducir el propelente y la tobera sin problema sé que este se mueva internamente.

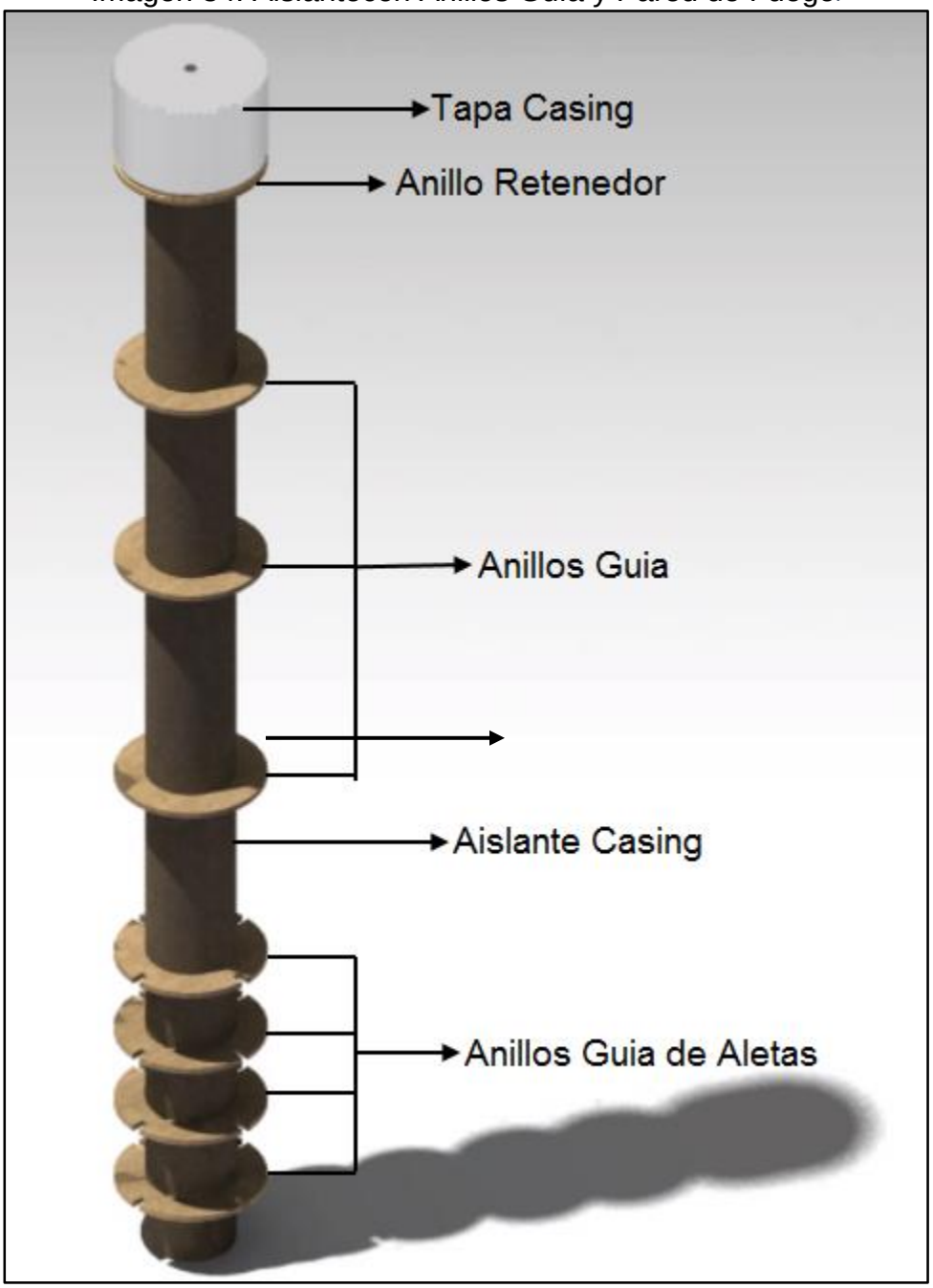
Imagen 33. Corte transversal de Cámara de Combustión y Anillo Seeger.



Fuente: Autores

Posteriormente la Cámara de Combustión, en su parte externa posee un aislante de cartón, el cual va fijado directamente al fuselaje, así permitiendo a la Cámara de Combustión extraerse cuando se requiere (Ejemplo: introducir el propelente), este cartón aloja unos anillos los cuales cumplen diferentes funciones, los primeros 4 anillos en la parte inferior alinean En la Cámara de Combustión y permite una fijación con el fuselaje y las aletas, adicionalmente, los siguientes 4 anillos, permiten seguir alineando en la Cámara de Combustión con respecto al fuselaje, el ultimo anillo de la parte superior permite de cierto modo detener la tapa de la Cámara de Combustión (Anillo Retenedor), y posteriormentese encuentra la pared de fuego ubicada encima de la tapa de la Cámara de Combustión, con un agujero en su centro, el cual pasa una argolla cáncamo.Como se puede observar en la siguiente imagen los diferentes elementos anteriormente mencionados.

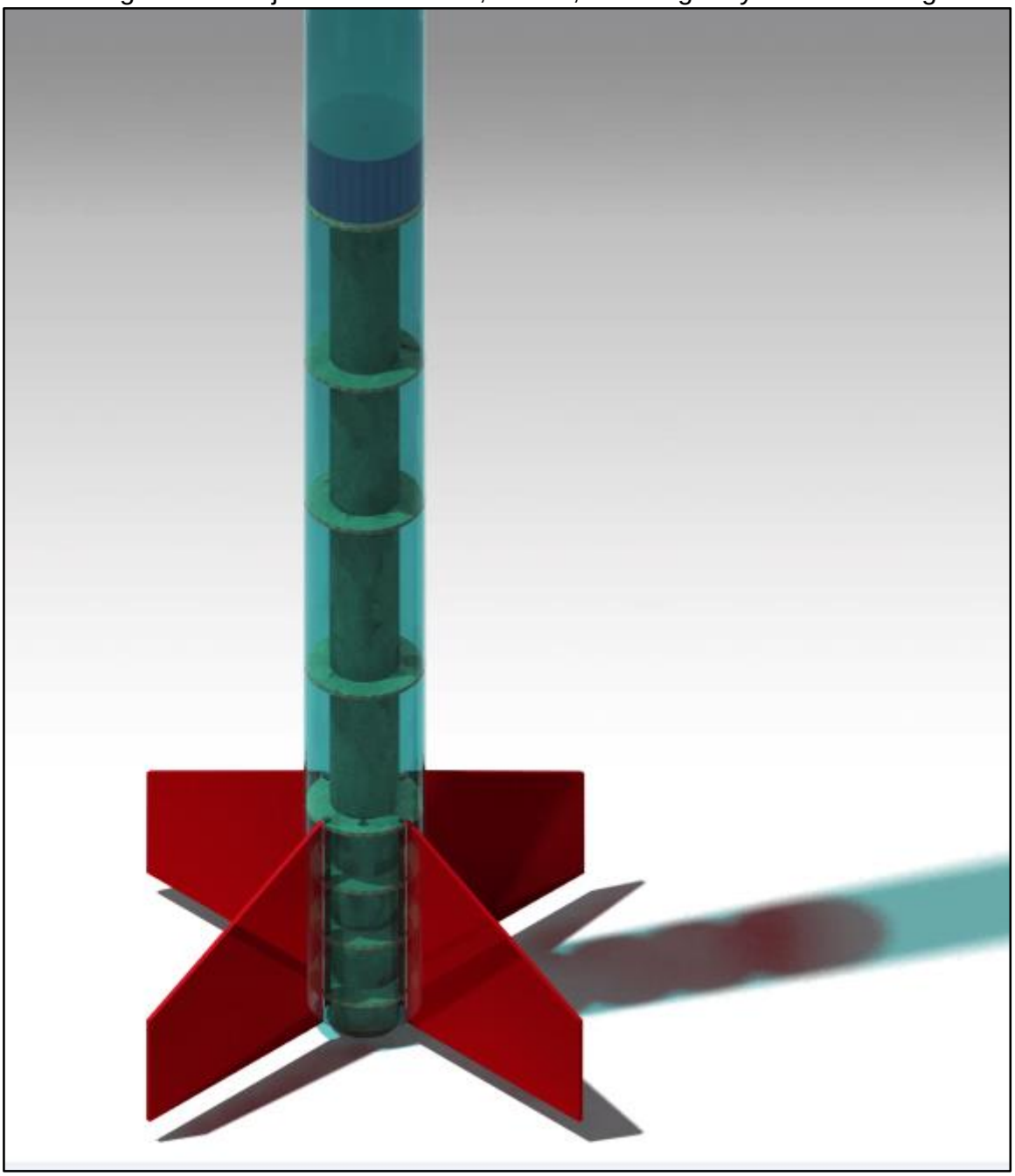
Imagen 34. Aislantecon Anillos Guía y Pared de Fuego.



Fuente: Autores

Finalmente, los anillos guías de las aletas son los mismos que se pueden observar en la imagen 34, adicionalmente se observan los anillos y la pared de fuego (Ver Anexos L, R), los cuales se pueden observar cómo es su conjunto con fuselaje y aletas en la siguiente imagen.

Imagen 35. Conjunto de Aislante, Aletas, Anillos guía y Pared de fuego.



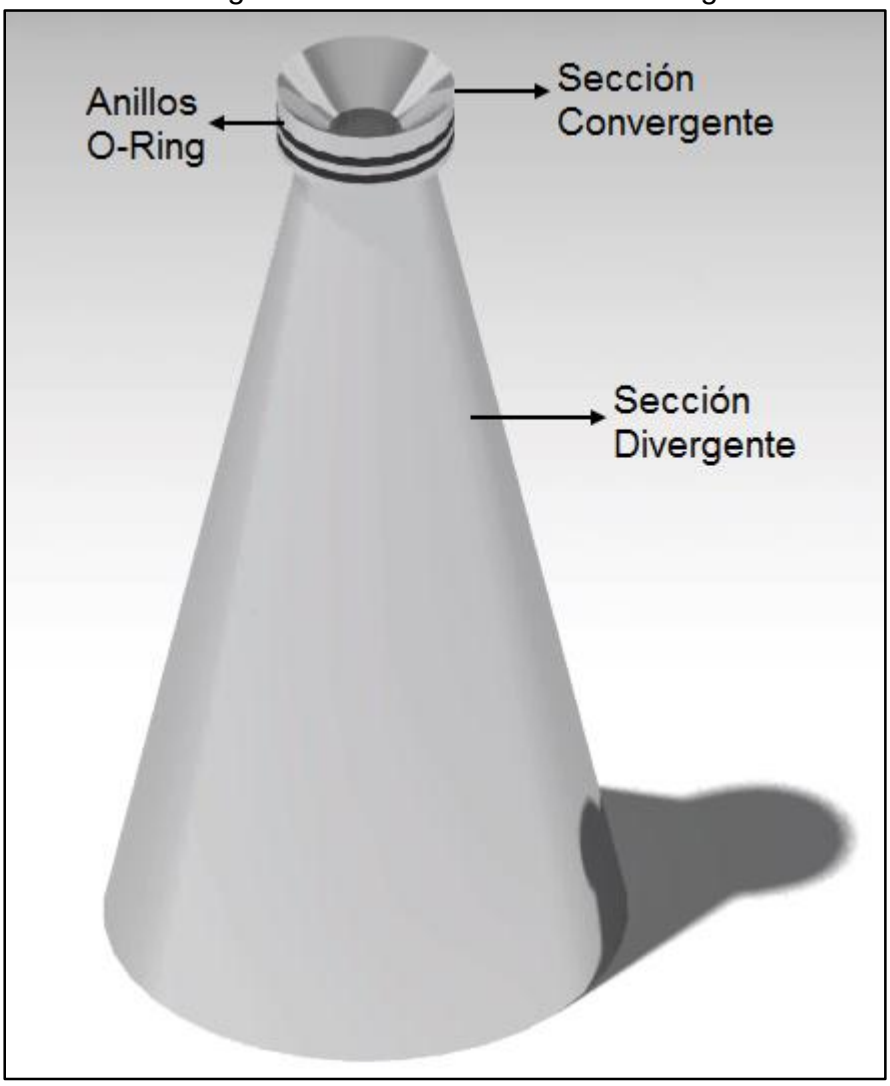
Fuente: Autores

3.1.5.2. Tobera

De acuerdo a los datos obtenidos en la sección 2.5. y en la Tabla 8 se realiza el diseño de la tobera para el cohete ARISTARCO I, se tiene en cuenta que hay un

espesor de material acorde (Ver Anexo N), también como se observa en la Imagen 36 la tobera tiene externamente dos anillos O-Ring en su sección convergente, cumpliendo la función de disminuir la perdida de presiones y generar un mejor ajuste con la pared interna de la Cámara de Combustión.

Imagen 36. Tobera con Anillos O-Ring



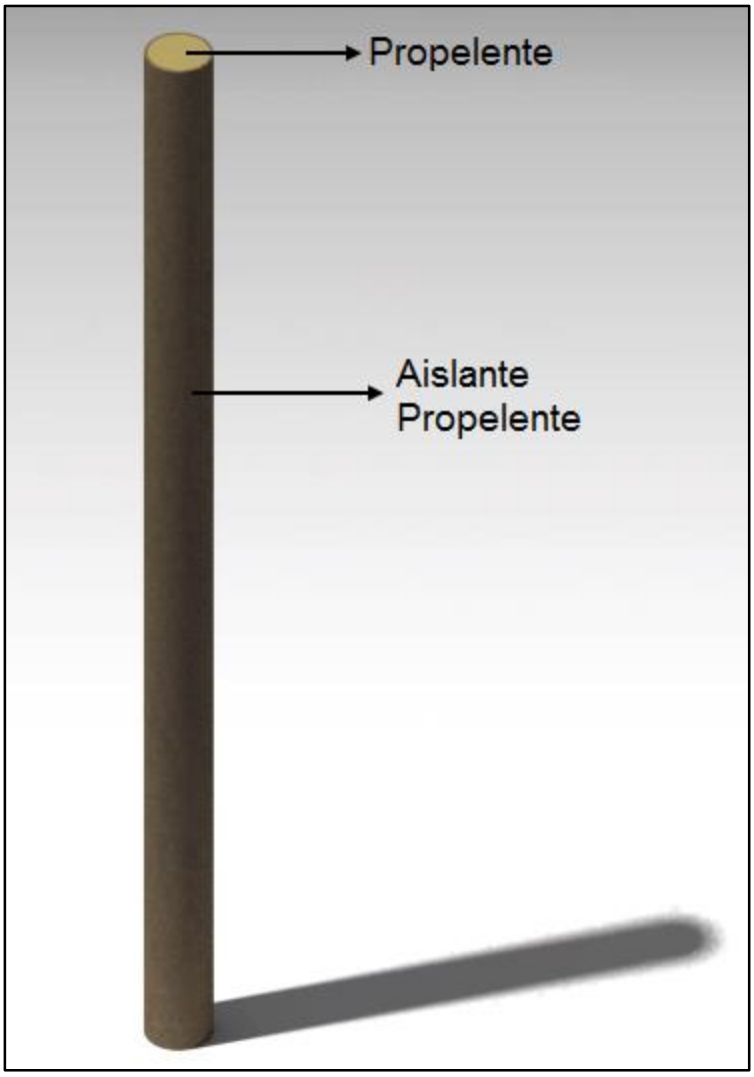
Fuente: Autores

3.1.5.3. Propelente

El propelente contiene una longitud total de 60 cm, el cual se encontrará envasado en un aislante cartón que posee un espesor de 0.34cm y el cual encaja en la Cámara de Combustión(Ver Anexo O).

Como se puede observar en la siguiente Imagen, el propelente dentro del cartón el cual cumple función de aislante, y facilita su adecuación a la Cámara de Combustión.

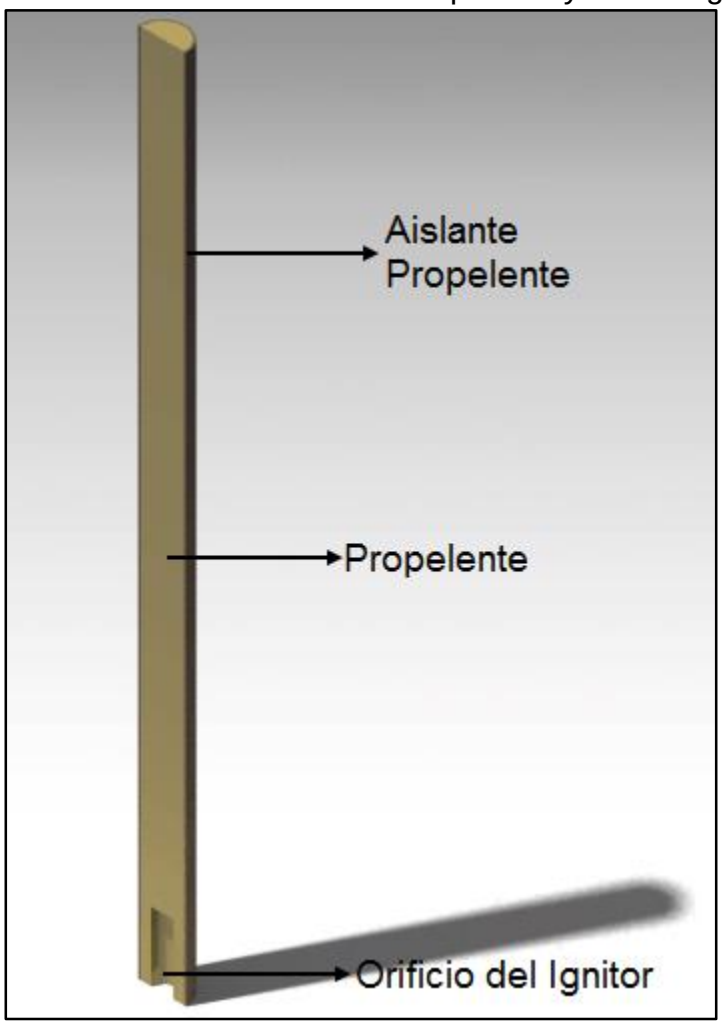
Imagen 37. Vista Isométrica del Propelente envasado en el Aislante



Fuente: Autores

Dicho propelente por las características anteriormente mencionadas posee un orificio adecuado para poner el ignitor para su posterior combustión, como se puede notar en la siguiente imagen.

Imagen 38. Corte transversal del Propelente y Orificio Ignitor.



Fuente: Autores

El propelente junto con su aislante debe encajar en una cámara de combustión acorde para realizar un quemado evitando el escape de presión de este por alguna sección diferente a la tobera, por lo tanto, se diseña la Cámara de Combustión para el Motor del Cohete ARISTARCO I.

3.2. Sistema de Recuperación.

El sistema de recuperación del cohete ARISTARCO I, en su estado plegado y con el sistema eyección ocupa aproximadamente un espacio de aproximadamente longitud de 25cm y diámetro de 6.8cm (Sección 2.3.3.11.1.1.), y este va ubicado

en la parte superior de la computadora de vuelo e inmediatamente por debajo de la ojiva, como se puede observar en la Imagen 39.

Imagen 39. Ubicación del Sistema de Recuperación, Ojiva y Carga Paga

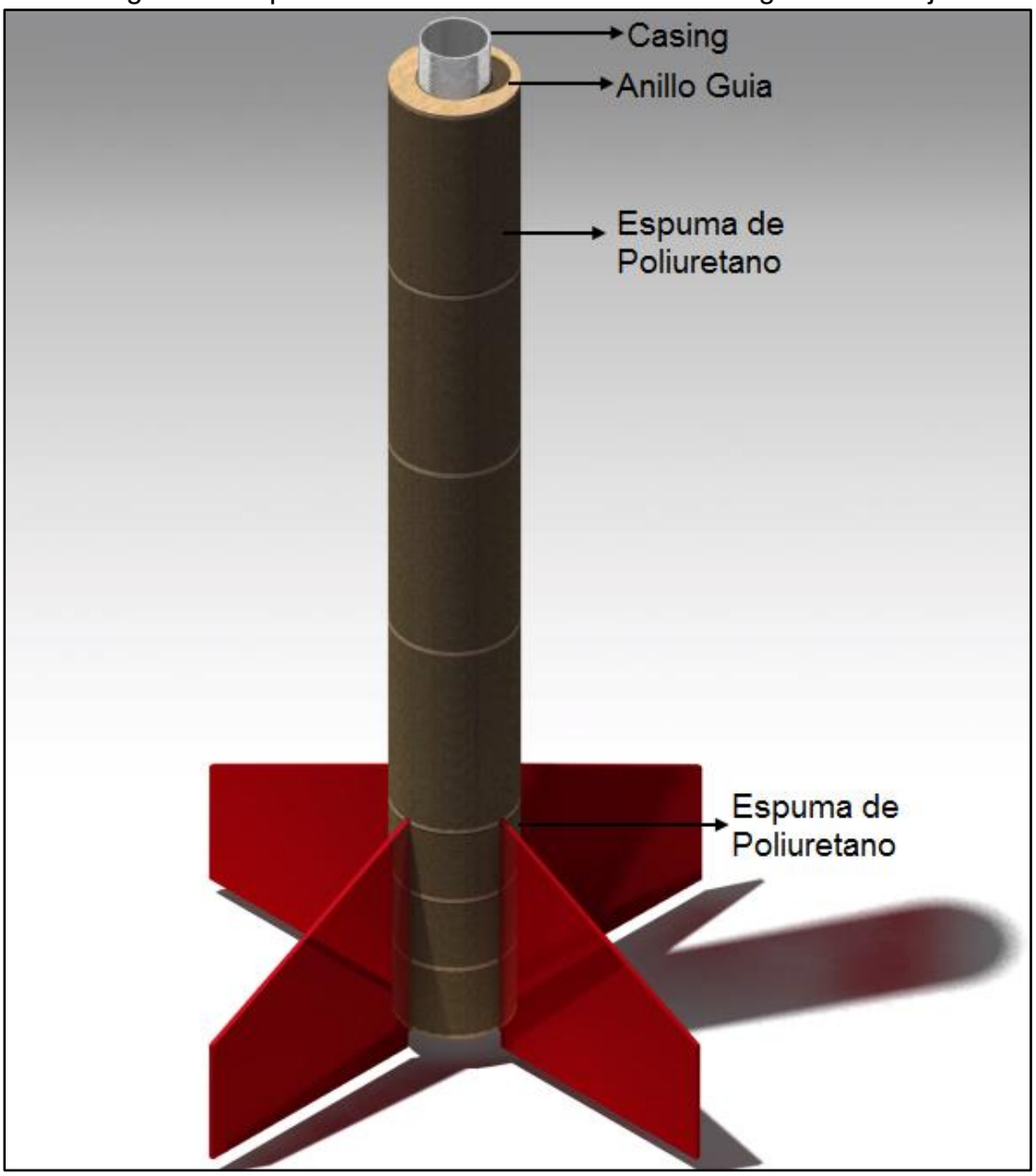
Fuente: Autores

3.3. Aislante Térmico

Como se puedo evidenciar en el Diseño Conceptual y Preliminar del cohete ARISTARCO I el aislante térmico usado para el cohete es la Espuma de Poliuretano, en la siguiente imagen se puede observar la ubicación de dicha espuma en el cohete ARISTARCO I.

En la Imagen 40 se puede observar la ubicación del aislante térmico a lo largo del soporte de la Cámara de Combustión, se puede ver que las tres secciones ubicadas en la sección de las aletas tienen una dimensión diferente a las otras 4 secciones posteriores, esto se da debido al número de anillos ubicados en las aletas y posterior a este, hay que aclarar que en la imagen se evidencia sin la estructura del fuselaje (Ver Anexos P, Q)

Imagen 40. Espuma de Poliuretano ubicado a lo largo del fuselaje.



Fuente: Autores

En la siguiente imagen se puede observar el aislante entre la pared de fuego y la computadora de vuelo, con una longitud total de 10.5cm, este aislante posee un agujero en su centro para la ubicación del cáncamo y así poder ajustarlo con la línea del paracaídas, como antes se ha mencionado (Ver Anexos M, P, Q).

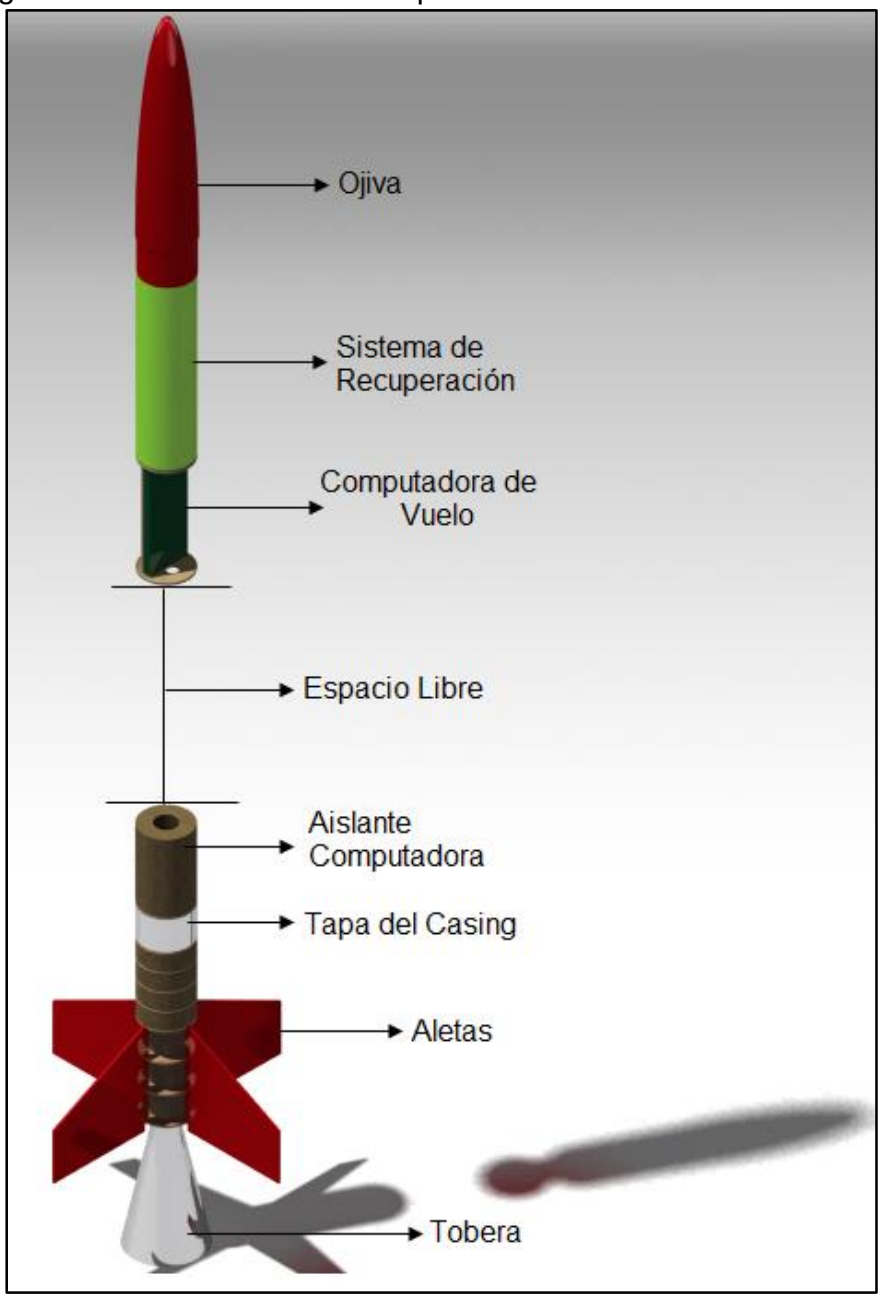
Imagen 41. Ubicación del Aislante de la Computadora.

Fuente: Autores

3.4. Masa de Lanzamiento

Posteriormente luego de haber detallado cada uno de los sistemas y componentes del cohete sonda ARISTARCO I, se puede observar en la siguiente imagen el acople de todos los elementos dentro del fuselaje del cohete sonda.

Imagen 42. Ubicación de los componentes internos del ARISTARCO I



Fuente: Autores

Por facilidades del diseño en CAD, el sistema de recuperación es representado con un cilindro el cual ocupa un similar espacio que el sistema de recuperación a usar durante la misión del cohete ARISTARCO I.

En la siguiente imagen se puede observar la ubicación del centro de gravedad y el centro de presiones a lo largo del cohete ARISTARCO I.

Imagen 43. Centro de Gravedad y Centro de Presiones en el ARISTARCO I.

Fuente: Autores

CONCLUSIONES

Se realizó el diseño conceptual del cohete sonda *ARISTARCO I,* en el cual, a partir de parámetros conceptuales, se evaluó cada uno de los componentes de forma cualitativa, con fin de determinar el tipo y características que definen los componentes que conforman el vehículo teniendo en cuenta la misión para la cual es diseñado el mismo. De esta forma, se definen los componentes y características que posee el cohete*ARISTARCO I,*en cada uno de sus sistemas (Sistema de propulsión, estructura, carga paga, sistema de recuperación y sistema de operación en tierra).

A través de una evaluación de carácter cuantitativo acerca de las características de operación que presenta dicho cohete, se realizó el diseño preliminar de un cohete sonda de propelente solido que realiza mediciones de condiciones atmosféricas por encima de un kilómetro de altitud, el cual tiene idealmente apogeo de 3,4Km sobre la superficie de lanzamiento en un periodo de tiempo de aproximadamente 26,739 segundos. El cohete diseñado, posee una masa total de lanzamiento de 1778,248gr, de la cual 364,17gr pertenece al propelente, cuyo valor equivale el 20,4% de la masa total de este. La masa de la carga paga que transporta el cohete es de aproximadamente 100gr, la cual equivale al 5,62% de la masa total de lanzamiento. De lo anterior, se tiene que la masa estructural del vehículo, incluyendo carga paga, corresponde al 77,88% de la masa de lanzamiento, equivalente a 1384,738gr.

El cohete sonda *ARISTARCO I,* alcanza una velocidad máxima de 256,09m/s, la cual equivale a un valor de 0,775 Mach y su motor genera un empuje máximo de 975,011 Newton.

El tiempo de quemado del propelente, es de 0,6344 segundos.

La presión a la que trabaja la cámara de combustión es de 8,425 MPa, con las dimensiones de motor establecidas en el presente documento.

Las dimensiones de la tobera se pueden optimizar, empleando menor cantidad de material, si el valor de la presión admisible en la cámara de combustión se aumenta, lo cual se logra cambiando el material a utilizar y teniendo en cuenta sus propiedades mecánicas.

A partir de las dimensiones de la carga paga que debía ser incorporada in el interior del ARISTARCO I, se seleccionó el diámetro del fuselaje (7,2cm), a partir del cual, mediante el rango aceptable de la relación longitud-diámetro, se determinó el valor de la longitud del fuselaje, cuyo valor es de 120cm. A partir de este y teniendo en cuenta la relación longitud-diámetro para la ojiva, se determinó la longitud de esta en 30cm, obteniendo así, la longitud total que posee el *ARISTARCO I,* cuyo valor es de 150cm.

A través de una revisión realizada, se definieron los sistemas principales que componen un cohete; de esta forma, se determinaron los principales sistemas que posee el *ARISTARCO I* (Sistema de propulsión, estructura, carga paga, sistema de recuperación y sistema de operación en tierra).

- Sistema de propulsión, el cual está integrado por dos componentes fundamentales, la cámara de combustión y la tobera convergente-divergente.

- Estructura, que está comprendida por ojiva, fuselaje, aletas estabilizadoras, carga paga, sistema de recuperación y motor.

- Carga paga, la cual está constituida por un sensor de presión y temperatura (BMP-180), y una tarjeta MicroSD para almacenar los datos obtenidos.

- Sistema de recuperación, integrado por un sensor que permite obtener la inclinación del cohete (MPU-6050), y dependiendo de dicha inclinación, activar dos motores encargados de expulsar el paracaídas.

- Operación en tierra, cuyo sistema está comprendido por estopín, detonador, control remoto y localizador.

Se realizó el dimensionamiento y disposición de cada uno de los componentes del cohete *ARISTARCO I*, sobre el mismo. En el interior del fuselaje del cohete, a 1cm desde la base del mismo, se encuentra posicionado el sistema de propulsión (SP) con una longitud de 25cm, seguido de este, a 28cm de la base del vehículo, se encuentra ubicado el aislante térmico de la carga paga (AisPL) con una longitud de 10.5cm, posterior a este, a 77cm de la base del ARISTARCO I, con longitud de 15cm se encuentra la carga paga (PL) y a continuación, es decir a 92cm de la base del cohete, está ubicado el sistema de recuperación (SR), con longitud de 25cm.

RECOMENDACIONES

- Se recomienda para un posterior trabajo de grado realizar el análisis aerodinámico del cohete sonda “ARISTARCO I” y así obtener el comportamiento detallado del vehículo durante el vuelo.

- Para implementación de cada uno de los componentes del “ARISTARCO I” ya fabricados es necesario revisar el estado actual del Banco de Pruebas de motor cohete de la Fundación Universitaria Los Libertadores, para realizar las pruebas necesarias y así observar el comportamiento de este en pruebas estáticas.

- Se aconseja realizar un estudio detallado de la trayectoria de vuelo de acuerdo a las características particulares del cohete sonda “ARISTARCO I”

- Realizar un estudio detallado de transferencia de calor del motor cohete del “ARISTARCO I”, obteniendo parámetros de gran relevancia para tener en cuenta al momento de la construcción del cohete.

- Se recomienda diseñar diferentes tipos de toberas de acuerdo a misiones particulares mediante la variación del material de la cámara de combustión, que también puede cumplir el cohete “ARISTARCO I” sin cambiar sus demás dimensiones de los distintos componentes.

- Diseñar y realizar el procedimiento adecuado para la construcción del propelente solido tipo Candy Amateur garantizando el mayor rendimiento de este a la hora de quemado teniendo en cuenta las dimensiones correspondientes al “ARISARCO I”

- Diseñar y construir una plataforma de lanzamiento óptima para ejecutar la misión del “ARISTARCO I”.

- Se recomienda hacer una optimización estructural a través de nuevos materiales disponibles en la industria aeroespacial para un mejor rendimiento del cohete “ARISTARCO I”.

- Realizar un protocolo con las medidas de seguridad y reglamentos nacionales e internacionales necesarios para el lanzamiento del cohete “ARISTARCO I”.

- Se recomienda utilizar la fracción de masa que se tiene como lastre para implementar carga paga adicional a la computadora de vuelo de cohete “ARISTARCO I”.

REFERENCIAS

Aislamientos Saiz Martínez. El aislamiento y el PUR. Características y aplicaciones. Recuperado el 13 de julio de 2016, de: http://www.saizmartinez.es/pdf/caracteristicas-aplicaciones.pdf

ALACER MAS. PROPIEDADES MECANICAS TIPICAS ALUMINIO 7075. Recuperado el 15 de agosto de 2016, de: http://www.alacermas.com/img/galeria/files/aluminio/chapa_7075_aluminio.pdf.

American Pyrotechnics Association. Glossary of Pyrotechnic Terms. Recuperado el 13 de julio de 2016, de: http://www.americanpyro.com/glossary-of-pyrotechnic-terms

Brito. Jean Frank. DISEÑO Y EVALUACION CONCEPTUAL DE UN MOTOR COHETE DE COMBUSTIBLE SOLIDO. Sartenejas. 2011.

Calcular Área. Calcular Área del Circulo. Recuperado el 16 de agosto de 2016, de: http://www.calculararea.com/circulo.htm.

Calcular Área. Calculo de la Superficie del Trapecio. Recuperado el 9 de agosto de 2016, de: http://www.calculararea.com/trapecio.htm.

Castillo, Leudy. Jiménez Nixon. DISEÑO, DESARROLLO Y PRUEBAS DEL SISTEMA DE RECUPERACION PARA EL COHETE SONDA LIBERTADOR I. Bogotá D.C, 2014.

Coast Pink. Aerojet General X-8. Recuperado el 06 de Julio de 2016, de: http://coast.pink/aerojet-general_261917.html.

Coast Pink. Berenice. Recuperado el 06 de Julio de 2016, de: http://coast.pink/berenice-cohete_632520.html.

Cohetería experimental amateur CEA, Las aletas estabilizadoras. Recuperado el 5 de agosto de 2016, de:
http://www.angelfire.com/scifi2/coheteria/aletas/aletas.htm
Correal. M, Castaño. A, Rincón. O y Aguillón. H. DISEÑO Y CONSTUCCION DE UN COHETE AMATEUR TIPO G. Bogotá. 2009.

De León. Pablo. HISTORIA DE LA ACTIVIDAD ESPACIAL ARGENTINA. Argentina. 2008.

Díaz. Otoniel. Construcción de Cohetes de Baja Potencia. Proyecto MSEIP-Engineering INTER-Bayamón. 2015.

DIGITAL TRENDS. Trackerpad. Recuperado el 10 de junio de 2016, de: http://www.digitaltrends.com/cool-tech/trackerpad-gps-sticker-kickstarter/.

DIYhacking. ARDUINO MPU 6050. Recuperado el 20 de agosto de 2016, de: http://diyhacking.com/arduino-mpu-6050-imu-sensor-tutorial/.

E. G. Ewing, H.W. Bixby y T.W. Knache. Recovery system design guide. Gardena, California, 1978.

Editorial Norma. Enciclopedia Temática Ilustrada. La Aventura del Conocimiento.

ELECTRONICA PRACTICA APLICADA. SENSOR MPU-6050. Recuperado el 10 de junio de 2016, de: http://www.diarioelectronicohoy.com/blog/sensor-mpu-6050.

ELECTRONILAB. Sensor BMP-180. Recuperado el 10 de junio de 2016, de: http://electronilab.co/tienda/sensor-de-presion-barometrica-bmp180/.

Embedded Lab. USING BMP180 FOR TEMPERATURE, PRESSURE AND ALTITUDE MEASUREMENTS. Recuperado el 20 de agosto de 2016, de: http://embedded-lab.com/blog/bmp180/.

Encyclopedia Astronáutica. Arcas. Recuperado el 06 de julio de 2016, de: http://astronautix.com/lvs/arcas.htm.

Encyclopedia Astronáutica. Asp. Recuperado el 06 de julio de 2016, de: http://astronautix.com/lvs/asp.htm.
Encyclopedia Astronáutica. Astrobee. Recuperado el 06 de julio de 2016, de: http://www.astronautix.com/lvs/astrobee.htm.

Encyclopedia Astronáutica. Black Brant. Recuperado el 06 de julio de 2016, de: http://astronautix.com/lvs/blabrant.htm.

Encyclopedia Astronáutica. Cajun. Recuperado el 06 de julio de 2016, de: http://www.astronautix.com/lvs/cajun.htm.

FísicaBásica. Centro de Gravedad. Recuperado el 14 de agosto de 2016, de: http://www.geocities.ws/davidfisica/centrog.html.

Fortescue, P., Stark, J., & Swinerd, G. (2003). *SPACECRAFT SYSTEMS ENGINEERING* (3rd ed.). Londres.

Fuselaje, tecnología de materiales. Recuperado el 08 de mayo de 2016. De http://tecnologiadematerialesitesmgda.blogspot.com.co/

Gallardo Daniel. Apuntes de ARDUINO. Recuperado el 10 de junio de 2016, de: http://educacionadistancia.juntadeandalucia.es/profesorado/pluginfile.php/2882/mod_resource/content/1/Apuntes_ARDUINO_nivel_PARDILLO.pdf.

Gary. THE DESCRIPTIVE GEOMETRY OF NOSE CONES. 1996.

Giraldo. Horacio. EPOC Diagnóstico y tratamiento integral. Editorial Medica Panamericana. Tercera Edición. Bogotá. 2008.

Gómez, Fabián. Leiva, Huindi. ANÁLISIS DEL RENDIMIENTO DEL PROPELENTE SOLIDO TIPO AMATEUR MEDIANTE BALLISTIC EVALUATION MOTOR (BEM) Y SELECCIÓN DE LA TOBERA MAS ADECUADA PARA SU USO EN EL COHETE SONDA LIBERTADOR I. Bogotá DC. 2015.

HCX USER MANUAL Versión 1.1. Recuperado el 13 de junio de 2016, de: http://aiaaocrocketry.org/AIAAOCRocketryDocs/SLI2011-2012/Manuals/G-Wiz_HCX_User_Manual_1dot1.pdf

Hotmath.com, Latus Rectum. Recuperado el 14 de junio de 2016, de: http://hotmath.com/hotmath_help/spanish/topics/latus-rectum.html.

Inés Cedeño-Física. Área momento de inercia. Recuperado el 14 de agosto de 2016, de: https://sites.google.com/site/inescedenofisica/momento-de-inercia/ejercicios-propuestos.

InvenSense. MPU-6000 and MPU-6050 Product Specification. Recuperado el 10 de junio de 2016, de: https://store.invensense.com/datasheets/invensense/MPU-6050_DataSheet_V3%204.pdf.

Juan Parczewski´s Amateur Experimental Rocketry WEB Site. Diseño de Motores Cohete en 10 Pasos. Recuperado el 16 de agosto de 2016, de:

http://jpcoheteria.com.ar/Calc10PasosI.htm.

M. Bandecchi & B. Melton, F. Ongaro. (1999). Concurrent Engineering Applied to Space Mission Assessment and Design. Agencia Espacial Europea, bulletin 99.

Marcano. V, Benitez. P, La Rosa. C, La Cruz. L, Parco. M, Ferreira. J, Andressen. R, Serra. A, Peñaloza. M, Rodriguez. L, Cardenas. J, Minitti. V y Rojas. J. PROGRESOS ALCANZADOS EN EL PROYECTO UNIVERSITARIO COHETE SONDA ULA. Venezuela. 2009.

Nakka. Richard. Teoría Sobre Motores Cohete de Propelente Solido. Traducido por Garofalo. Sebastian. Recuperado el 16 de agosto de 2016, de: http://www.nakka-rocketry.net/articles/teoria_de_los_motores_cohete.pdf.

NASA, Payload Systems. Recuperado el 13 de junio de 2016, de: https://spaceflightsystems.grc.nasa.gov/education/rocket/payload.html

NASA, Scientific Ballons. Columbia Scientific Balloon Facility. Recuperado el 18 de mayo de 2016, de http://www.csbf.nasa.gov/balloons.html

Unión Internacional de Comunicaciones. CARACTERÍSTICAS TÉCNICAS Y CRITERIOS DE CALIDAD DE LOS SISTEMAS DE RADIOSONDAS DEL SERVICIO DE AYUDAS A LA METEOROLOGÍA

NASA, The Drag Equation. Recuperado el 14 de junio de 2016, de: https://www.grc.nasa.gov/www/k-12/airplane/drageq.html

NASA. Determining Center of Gravity. Recuperad el 15 de agosto de 2016, de: https://spaceflightsystems.grc.nasa.gov/education/rocket/rktcg.html.

NASA. Determining Center of Pressure-cp. Recuperado el 12 de agosto de 2016, de: https://spaceflightsystems.grc.nasa.gov/education/rocket/rktcp.html.

NASA. Isentropic Flow. Recuperado el 16 de agosto de 2016, de: https://www.grc.nasa.gov/www/k-12/airplane/isentrop.html.

NASA. Nose Cone Volume. Recuperado el 8 de agosto de 2016, de: https://www.grc.nasa.gov/www/k-12/airplane/volume.html.

NAYLAMPS MECHATRONICS. Tutorial Arduino y memoria SD y micro SD. Recuperado el 20 de agosto de 2016, de: http://www.naylampmechatronics.com/blog/38_Tutorial-

Arduino-y-memoria-SD-y-micro-SD-.html.

OnlineMschol. Volumen del Cilindro. Recuperado el 8 de agosto de 2016, de: http://es.onlinemschool.com/math/assistance/figures_volume/cylinder/.

OnlineMSchool. Área de Cilindro. Recuperado el 14 de agosto de 2016, de: http://es.onlinemschool.com/math/assistance/figures_area_1/cylinder/.

OnlineMSchool. Volumen de la Prisma. Recuperado el 11 de agosto de 2016, de: http://es.onlinemschool.com/math/assistance/figures_volume/prism/.

Parks College Parachute Research Group, Calculating the descent rate of a round parachute, Recuperado el 19 de Julio de 2016, de: http://www.pcprg.com/rounddes.htm

PEREZ. E, BERMEO. N Y FÚQUEN. D. DISEÑO Y CONSTRUCCION DE UN MOTOR COHETE QUE PRODUZCA 800N DE EMPUJE. Bogotá. 2010.

Peterson Carl, Philip Hill. (1992). Mechanics and Thermodynamics of Propulsion (Second Edition).

Reboiras. M. QUIMICA: La ciencia básica. Editorial Thompson. 2008.

Recuenco. Jesus. MANUAL DEL CONSTRUCTOR DE MODELOS ESPACIALES. 2008.

Richard Nakka. Teoría sobre motores cohete de propelente sólido. Recuperado el 13 de junio de 2016, de: http://www.nakka-rocketry.net/

Richard Nakka's Experimental Rocketry Web Site. Burn Rate Determination from a Pressure-time Trace. Recuperado el 16 de agosto de 2016, de: http://www.nakka-rocketry.net/ptburn.html.

Richard Nakka's Experimental Rocketry Web Site, Parachute Design and Construction, Recuperado el 19 de Julio de 2016, de: http://www.nakka-rocketry.net/paracon.html

Richard Nakka's Experimental Rocketry Web Site. Rocket Motor Design Charts -Chamber Pressure-. Recuperado el 16 de agosto de 2016, de: http://www.nakka-rocketry.net/design1.html.

Richard Nakka's Experimental Rocketry Web Site. Rocket Motor Letter Code. Recuperado el 16 de agosto de 2016, de: http://nakka-rocketry.net/pix/class.gif.

Riveros Felipe, y Rodríguez Alejandro. Diseño y construcción de un cohete aficionado controlado mediante el accionamiento de una tobera de empuje vectorial. Universidad Militar Nueva Granada. Facultad de Ingeniería, 2010

Riveros. Felipe, Rodriguez. Luis. DISEÑO Y CONSTRUCCIÓN DE UN COHETE AFICIONADO CONTROLADO MEDIANTE EL ACCIONAMIENTO DE UNA TOBERA DE EMPUJE VECTORIAL. Bogotá. 2010.

Robledo, Jenny. IMPORTANCIA DEL DISEÑO Y CONSTRUCCION DE COHETES SONDA EN LATINOAMERICA, BRASIL Y ARGENTINA. Bogotá D.C. 2012.

ROBOT ROOM, Model Rocket Igniter Controller. Recuperado el 16 de junio de 2016, de: http://www.robotroom.com/Rocket-Ignition-System-1.html.

Science Learning, Rocket Aerodynamics. Recuperado el 26 de mayo de 2016. De http://sciencelearn.org.nz/Contexts/Rockets/Science-Ideas-and-Concepts/Rocket-aerodynamics

Primer Cohete-sonda argentino de dos etapas, Recuperado de: http://www.santafe-conicet.gov.ar/servicios/comunica/sonda.htm.

Science Learning, Rocket Aerodynamics. Recuperado el 7 de junio de 2016, de: http://sciencelearn.org.nz/Contexts/Rockets/Science-Ideas-and-Concepts/Rocket-aerodynamics.

Sistemas de Masa Variable. Movimiento Vertical de un Cohete. Recuperado el 17 de agosto de 2016, de: http://www.sc.ehu.es/sbweb/fisica/dinamica/cohete3/cohete3.html.

Vélez, Héctor. ESTUDIO AERODINÁMICO DE LA OJIVA DEL COHETE SONDA LIBERTADOR I MEDIANTE SOFTWARE CFD. Bogotá D.C, 2013.

Instructables. Design a Rocket Nose Cone.Recuperado el 20 de agosto de 2016, de: http://www.instructables.com/id/Design-a-Rocket-Nose-Cone-with-Software/?ALLSTEPS.

Fleeman. Eugene. Tactical Missile Design. American Institute of Aeronautics and Astronautics. 2001.

ANEXOS

ANEXO A. PROPIEDADES DEL PROPELENTE KN-SB (65/35)

Rocket Motor chamber pressure as a function of Kn KN.Dextrose (65/35) propellant		
R`	8314 J/mol-K	Universal gas constant
M	42.39 Kg/Kmol	Effective molecular wt of products
R	196.1 J7kg-K	Specific gas constant
K	1.043	Ratio specific heats. avg. value
η_c	0.95	Combustion efficiency
To	1710 K	Ideal combustion temperature
To act	1625 K	Actual chamber temperature
C^0	916 m/s	Characteristic exhaust velocity
$\acute{\rho}\ grain$	1.879 g/cm^3	Gain ideal density
$\rho/\acute{\rho}$	0.95	Density ratio (actual/ideal)
$\rho\ grain$	1.785 g/cm^3	Grain actual density

Tabla 1. Rocket Motor chamber pressure as a function of Kn KN.Dextrose (65/35) propellant.
Fuente: Rocket Motor Design Charts- Chamber Pressure -

ANEXO B. COEFICIENTES PARA LA DETERMINACIÓN DE KN

Rango de presiones	a'	b
2 MPa a 2,8 MPa	0	+ 78,57/MPa
2,8 MPa a 6,3 MPa	+ 164	+ 20/MPa
6,3 MPa a 8 MPa	+ 8,52	+ 44,68/MPa

Tabla 2. COEFICIENTES PARA LA DETERMINACIÓN DE KLEMMUNG
Fuente: Juan Parczewski´s Amateur Experimental Rocketry WEB Site. Diseño de Motores Cohete en 10 Pasos.

ANEXO C. INFORMACIÓN DE QUEMADO PARA PROPELENTE CANDY

KN-Dextrose									
Pressure range			a	n	Pressure range			a	n
psia			in/sec. (psia)		Mpa			mm/sec. (Mpa)	
15	to	113	0.016	0.619	0.103	to	0.779	8.88	0.619
113	to	373	0.311	-0.009	0.779	to	2.57	7.55	-0.009
373	to	860	0.005	0.688	2.57	to	5.93	3.84	0.688
860	to	1233	1.416	-0.148	5.93	to	8.50	17.2	-0.148
1233	to	1625	0.021	0.442	8.50	to	11.20	4.78	0.442

Tabla 3. INFORMACIÓN DE QUEMADO PARA PROPELENTE A BASE DE DEXTROSA.
Fuente: EFFECT OF CHAMBER PRESSURE ON BURNING RATE FOR THE POTASSIUM NITRATE - DEXTROSE AND POTASSIUM NITRATE - SORBITOL ROCKET PROPELLANTS

ANEXO D. TABLA DE COEFICIENTE DE EMPUJE DE COHETES

e Expansion area ratio

M_e: Mach number at exit plane

p_e/p_c: Ratio of pressure at exit plane to chamber pressure

$C_{f\text{-}opt}$: Thrust coefficient assuming optimal expansion

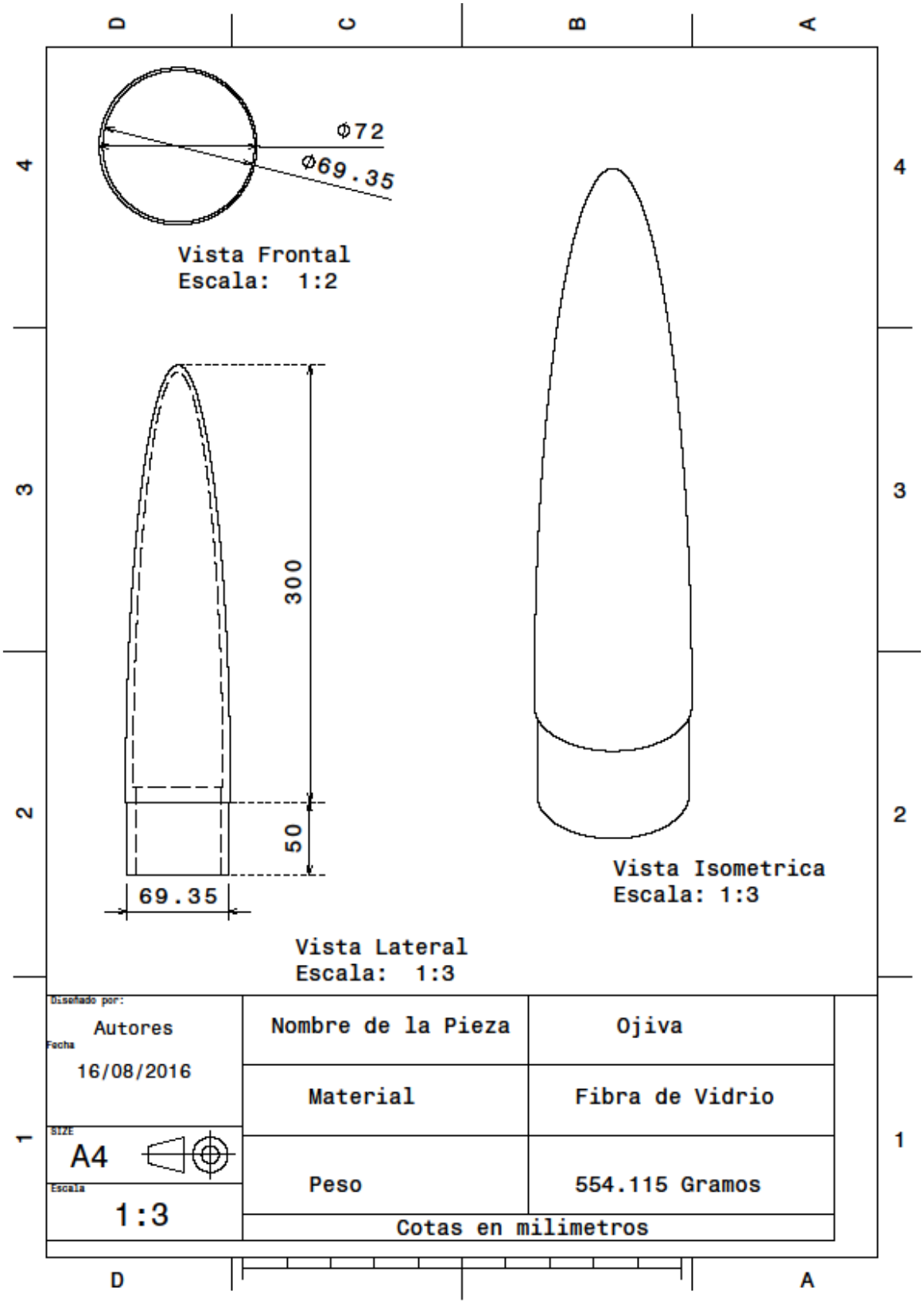
C_{fv}: Vaccuum thrust coefficient

epsilon	M_e	p_e/p_c	$C_{f\text{-}opt}$	C_{fv}
1.0	1.000	0.5643080	0.67753	1.24184
2.0	2.055	0.1207748	1.22420	1.46575
3.0	2.397	0.0656052	1.35714	1.55396
4.0	2.619	0.0435133	1.43312	1.60717
5.0	2.785	0.0319051	1.48482	1.64435
6.0	2.917	0.0248603	1.52331	1.67247
7.0	3.027	0.0201796	1.55359	1.69485
8.0	3.122	0.0168690	1.57833	1.71328
9.0	3.205	0.0144174	1.59911	1.72887
10.0	3.278	0.0125373	1.61693	1.74231
11.0	3.345	0.0110547	1.63247	1.75408
12.0	3.405	0.0098591	1.64620	1.76451
13.0	3.461	0.0088767	1.65847	1.77386
14.0	3.512	0.0080569	1.66952	1.78232
15.0	3.560	0.0073636	1.67957	1.79002

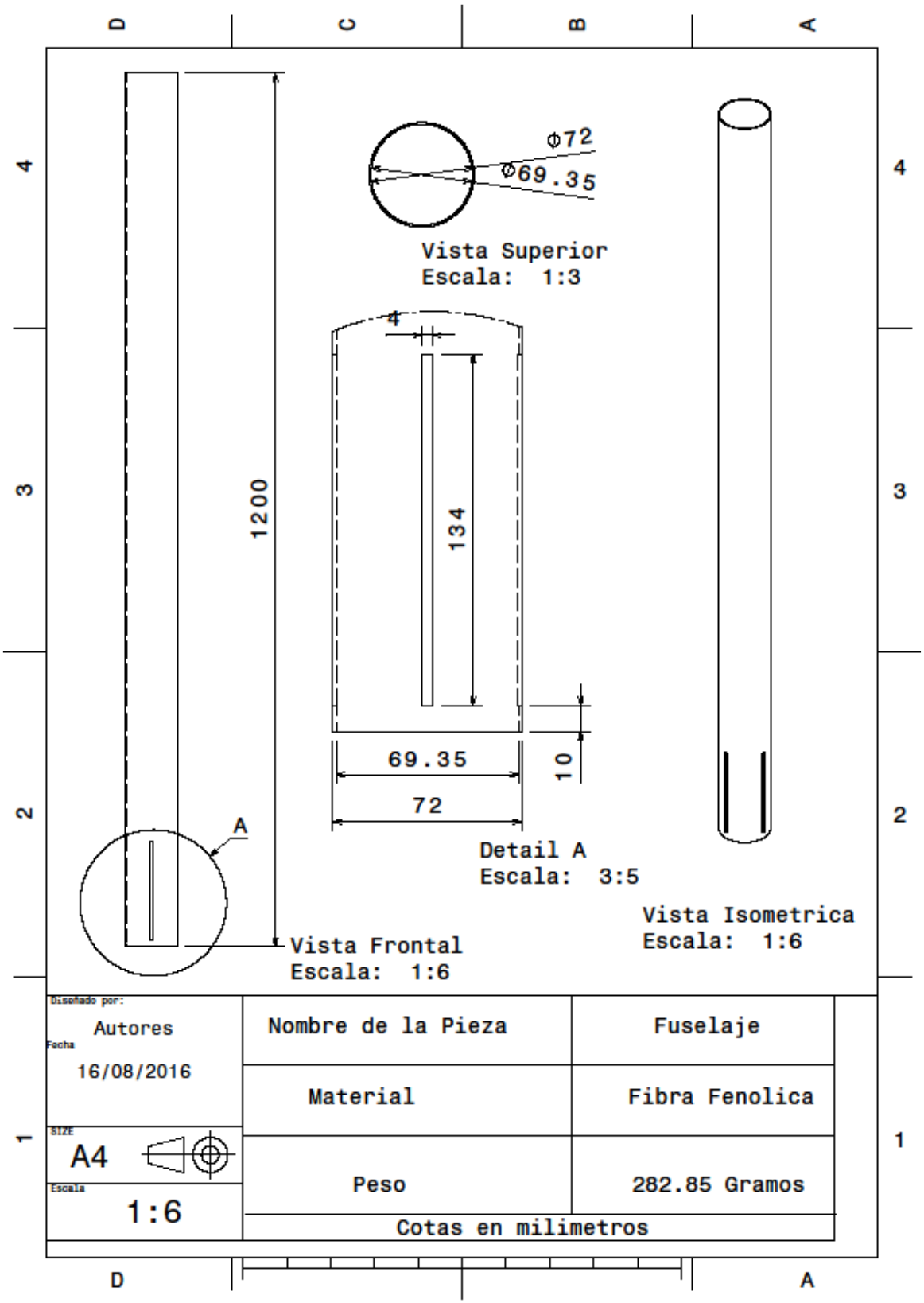
16.0	3.604	0.0067703	1.68875	1.79708
17.0	3.646	0.0062577	1.69721	1.80359
18.0	3.686	0.0058107	1.70502	1.80962
19.0	3.723	0.0054180	1.71229	1.81523
20.0	3.759	0.0050705	1.71906	1.82047

Fuente: PURDUE UNIVERSITY. Rocket Thrust Coefficient Tables. Disponible en: https://engineering.purdue.edu/Engr.

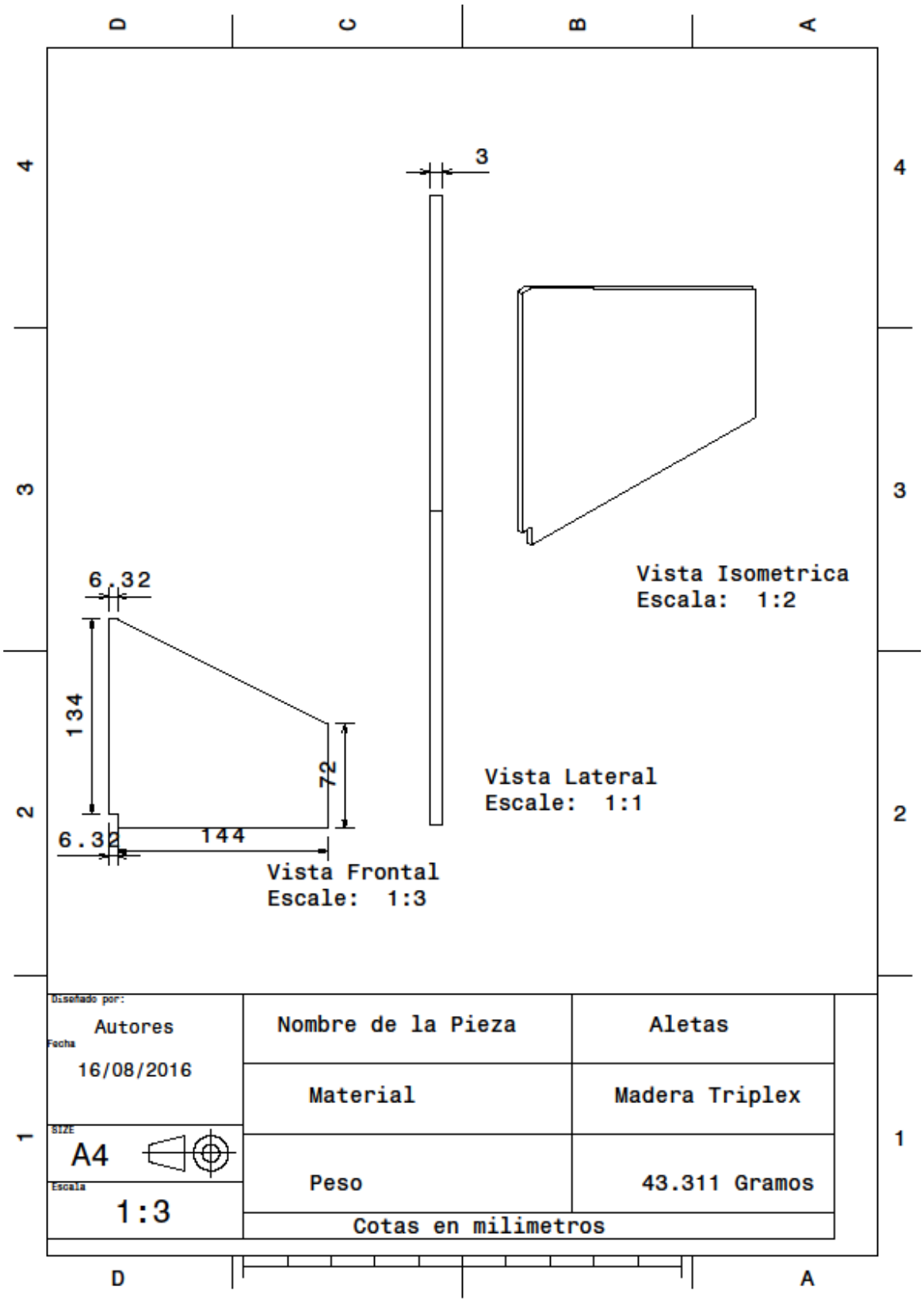
ANEXO E. OJIVA



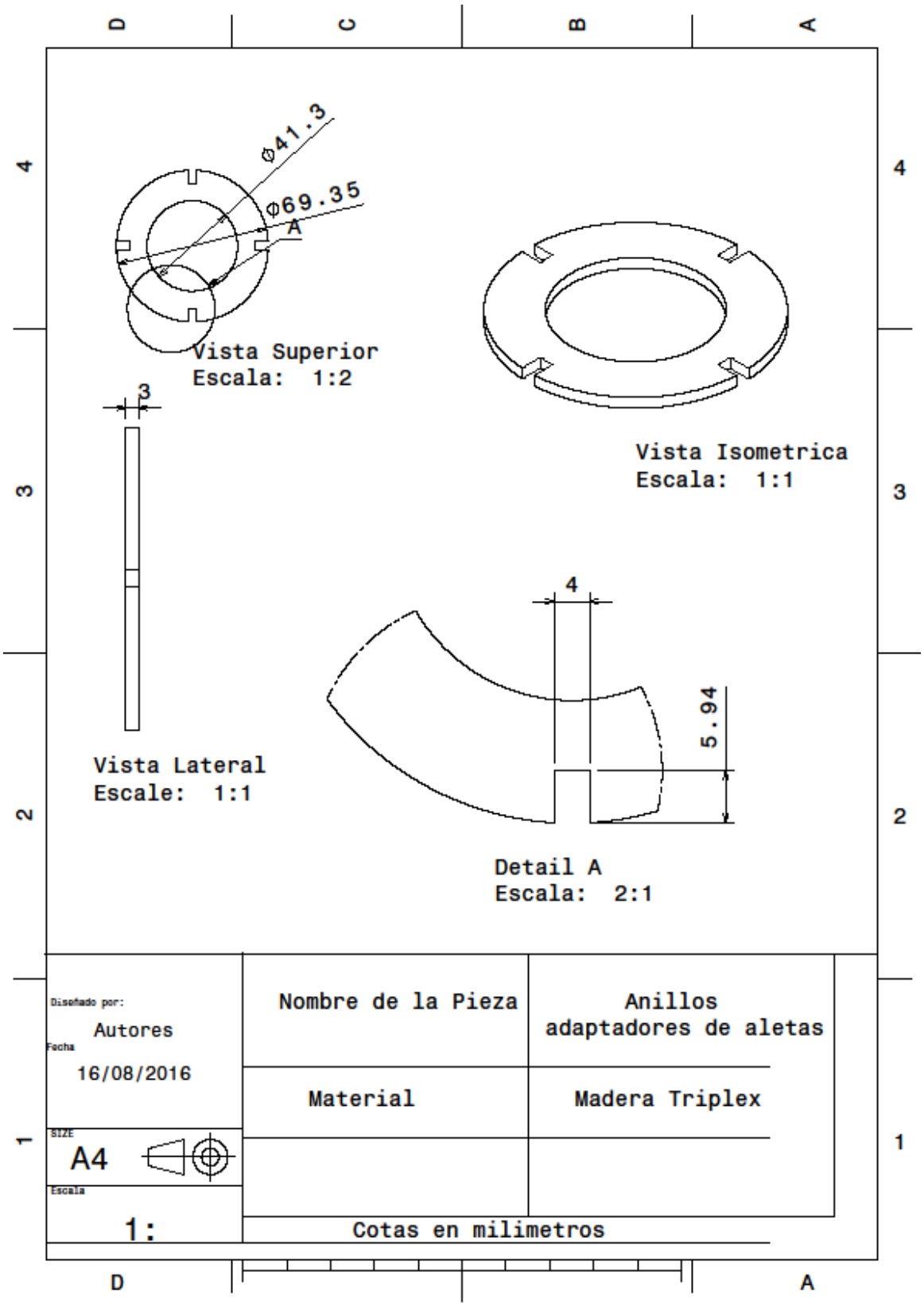
ANEXO F. FUSELAJE



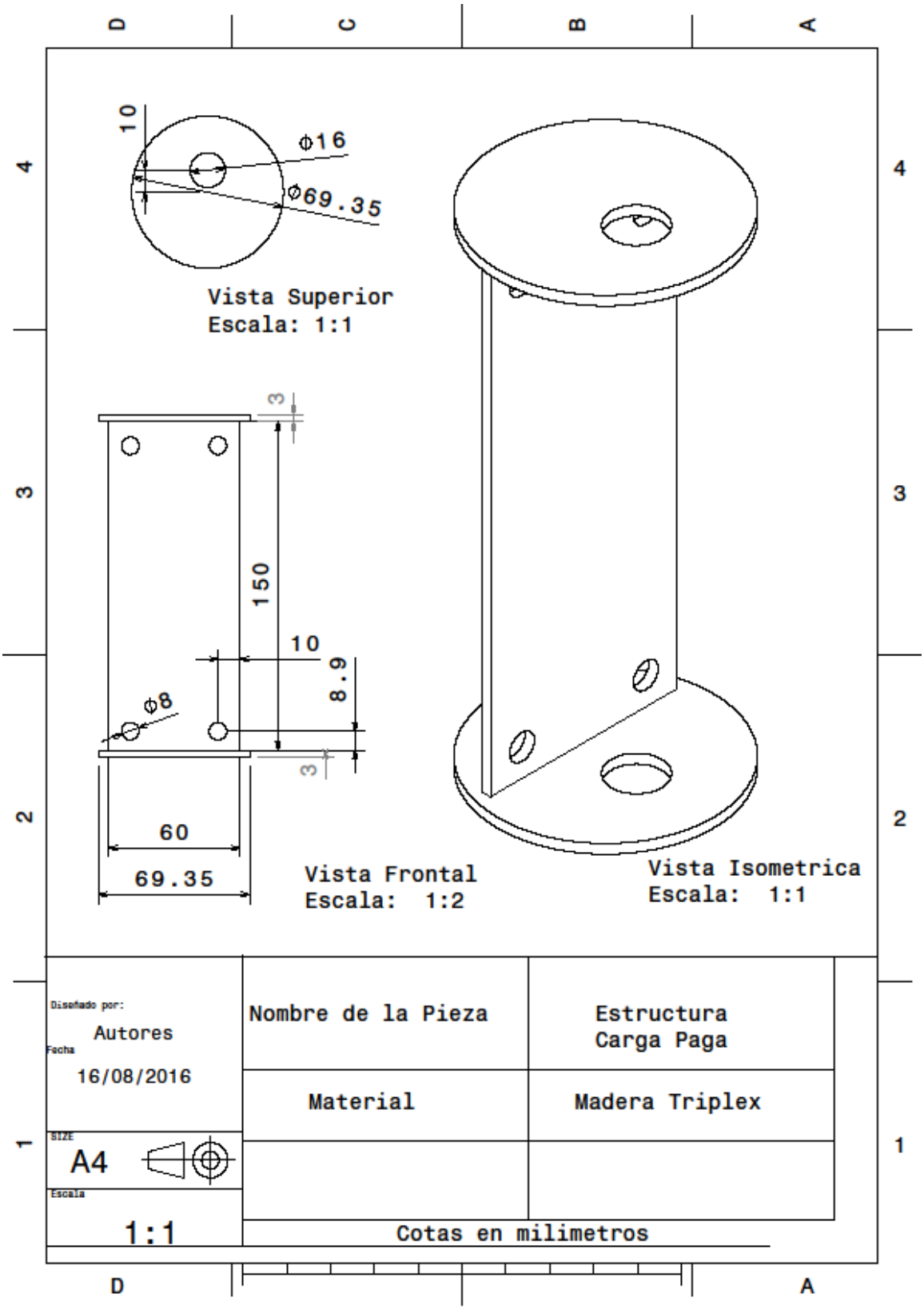
ANEXO G. ALETAS



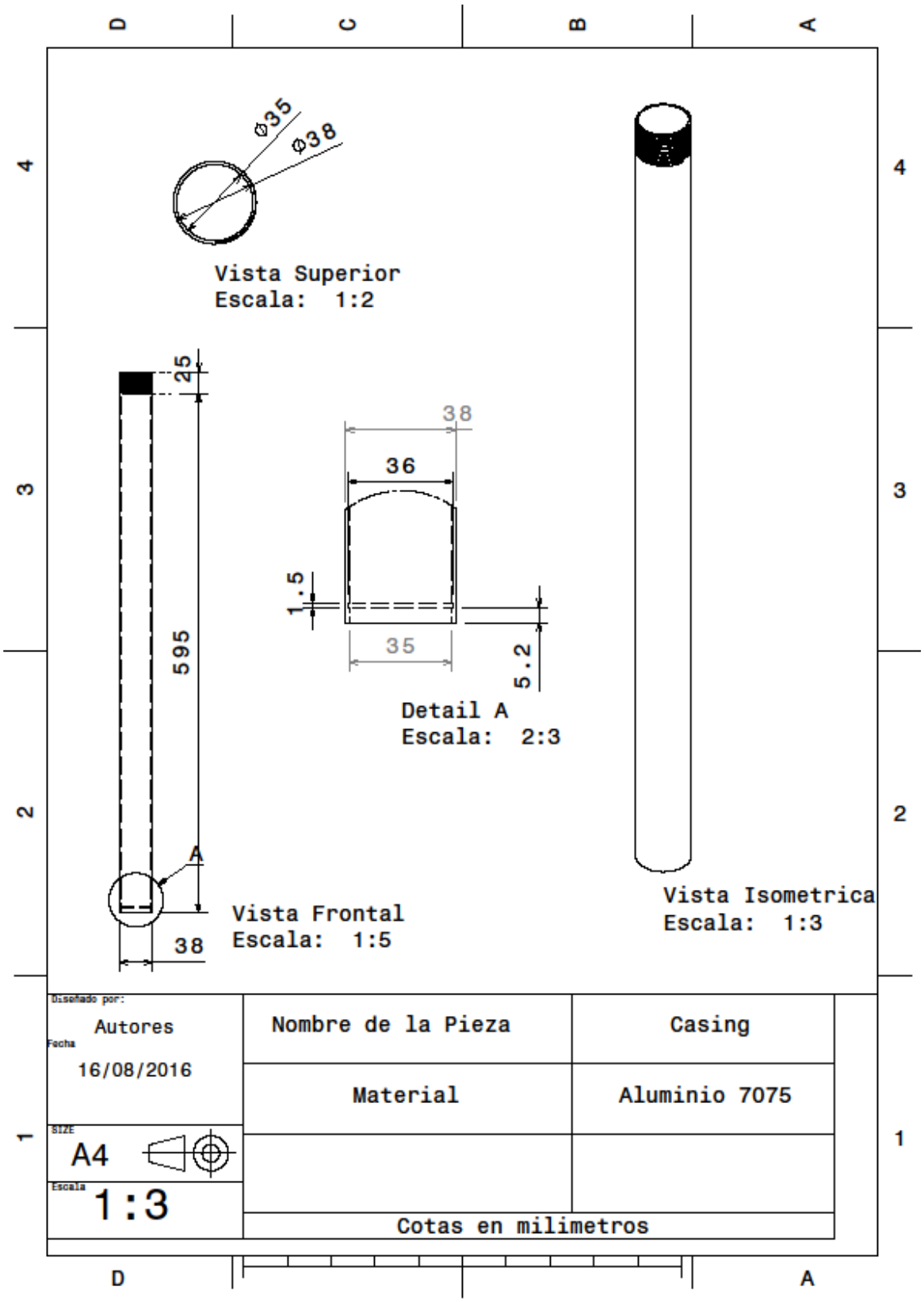
ANEXO H. ANILLOS ADAPTADORES DE ALETAS



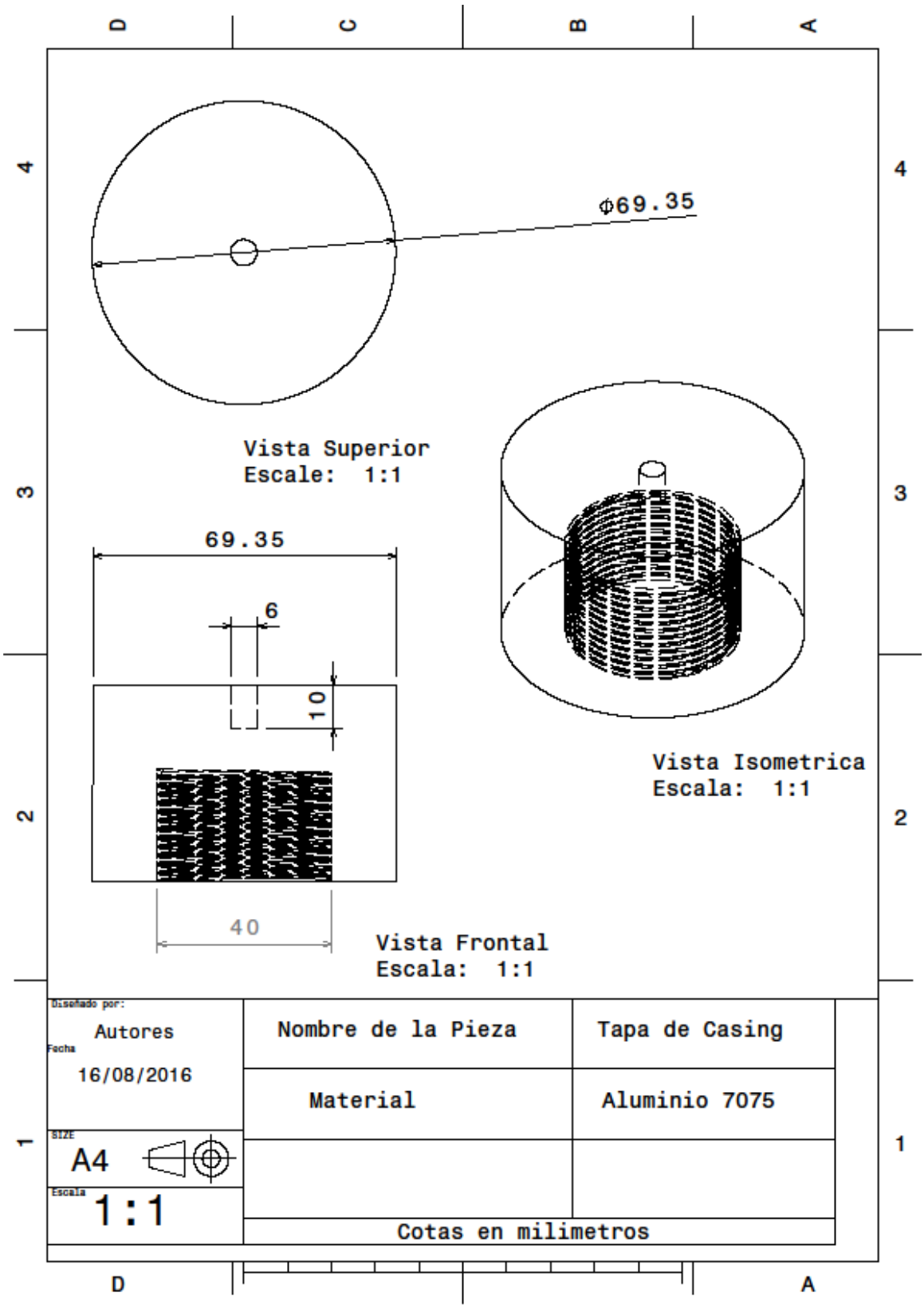
ANEXO I. ESTRUCTURA CARGA PAGA



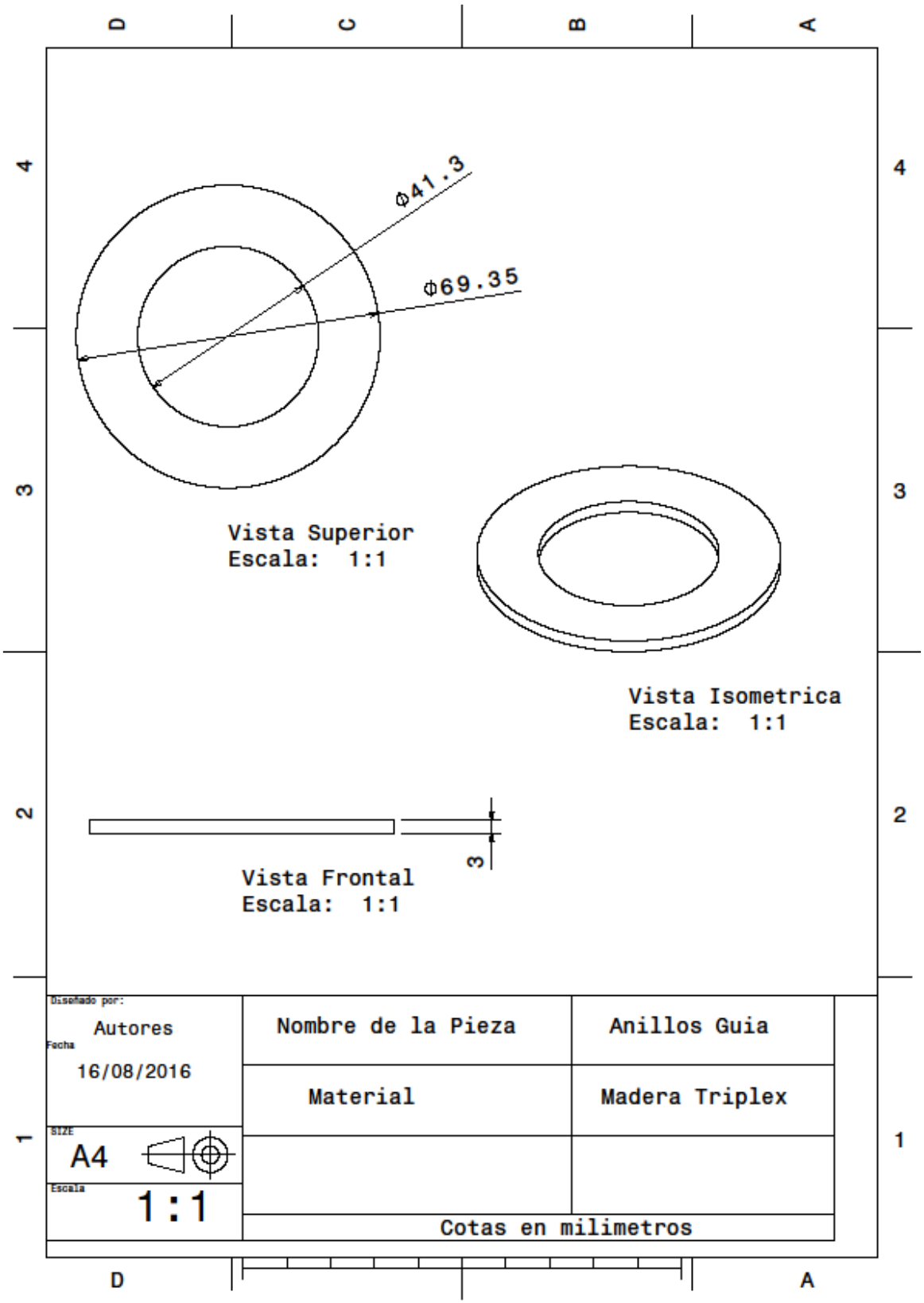
ANEXO J. CÁMARA DE COMBUSTIÓN



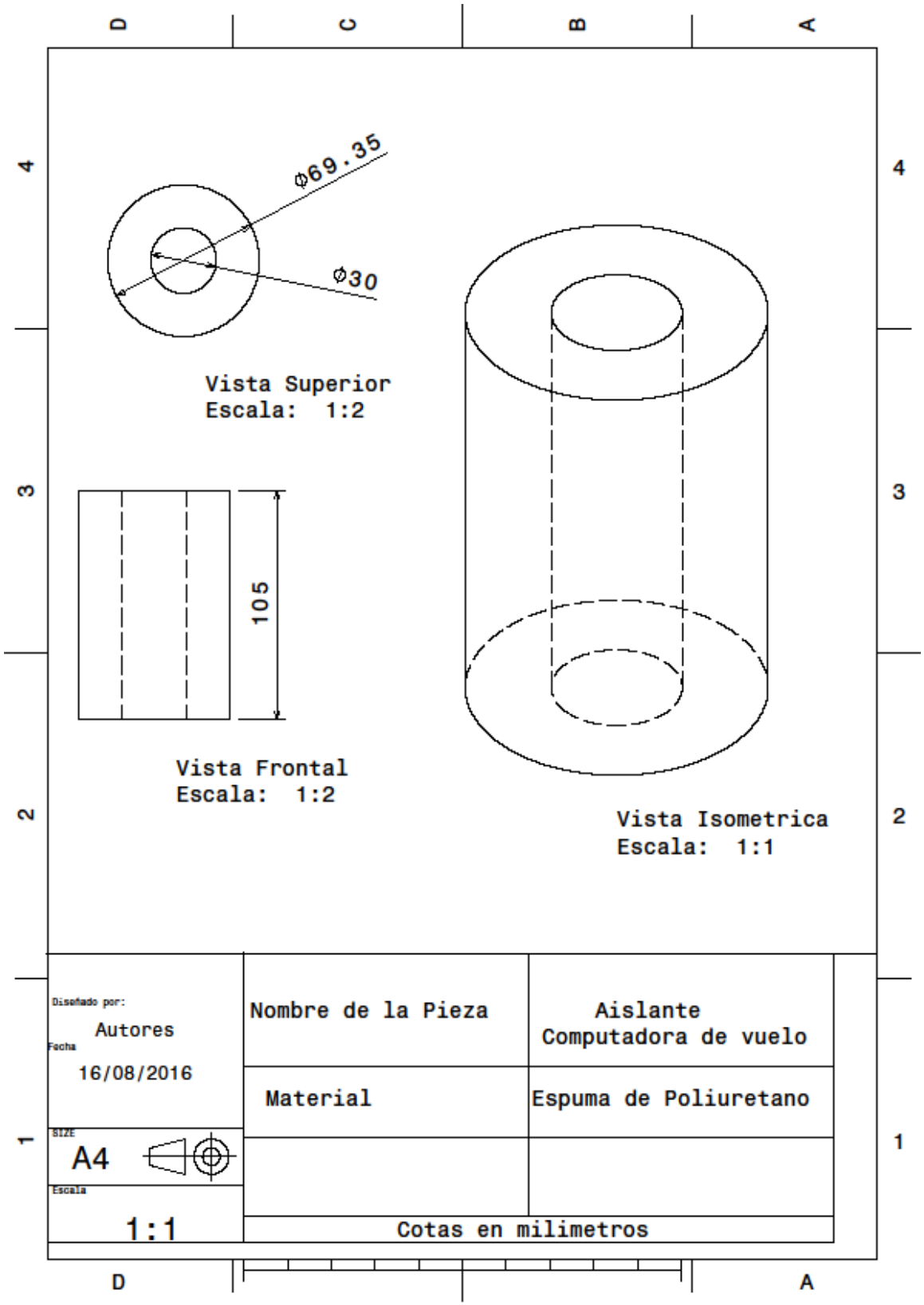
ANEXO K. TAPA DE CÁMARA DE COMBUSTIÓN



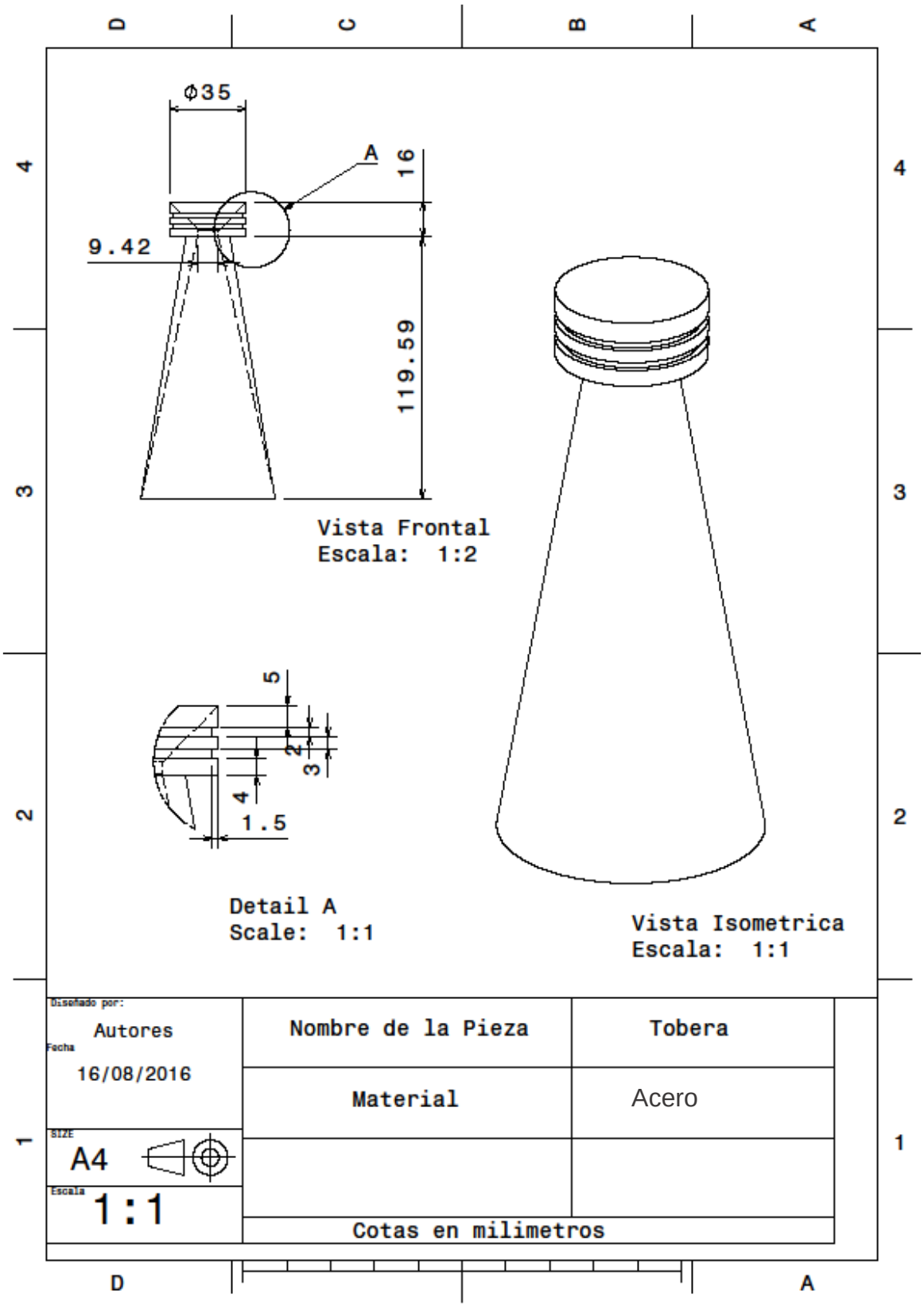
ANEXO L. ANILLOS GUIA



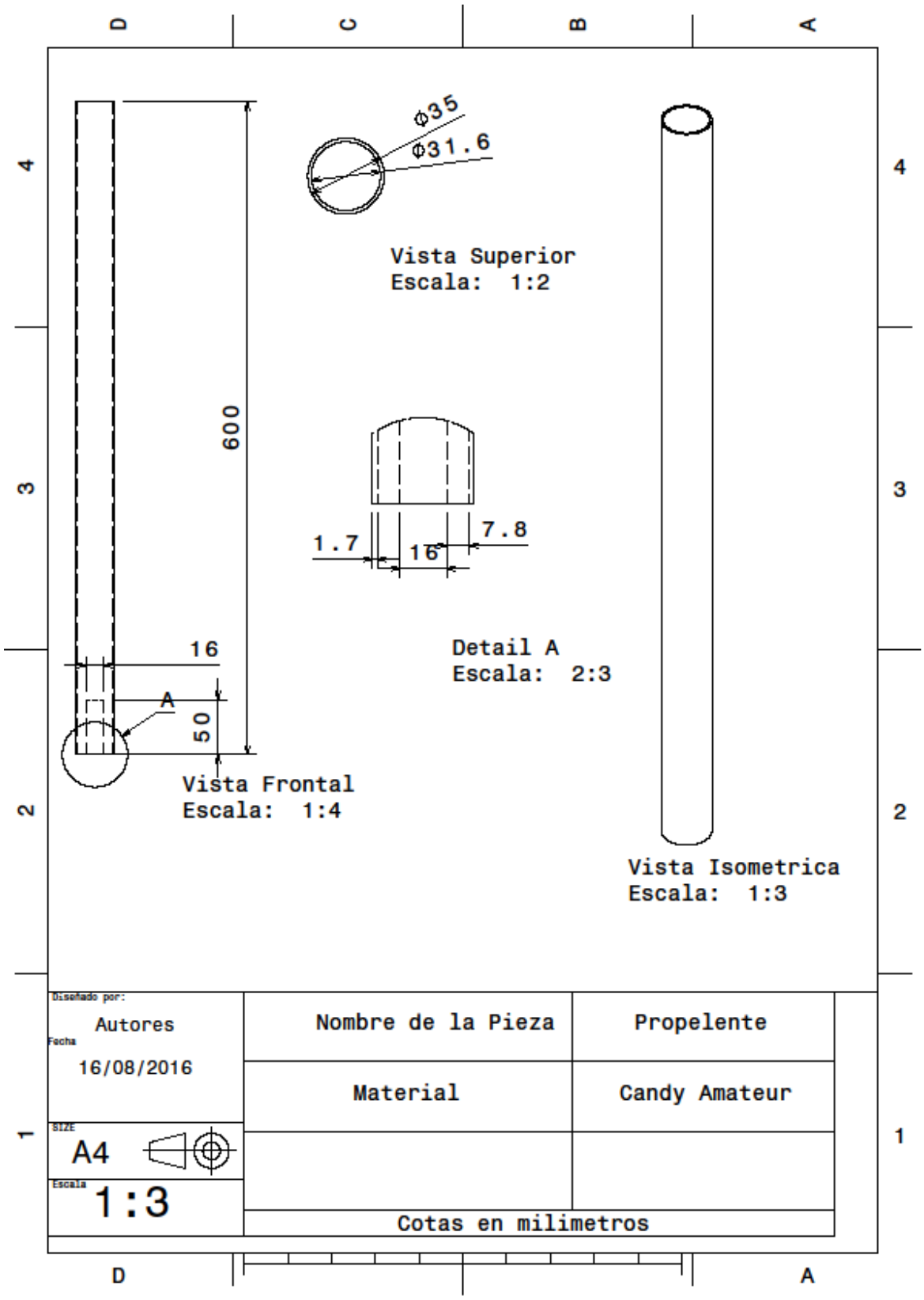
ANEXO M. AISLANTE COMPUTADORA DE VUELO



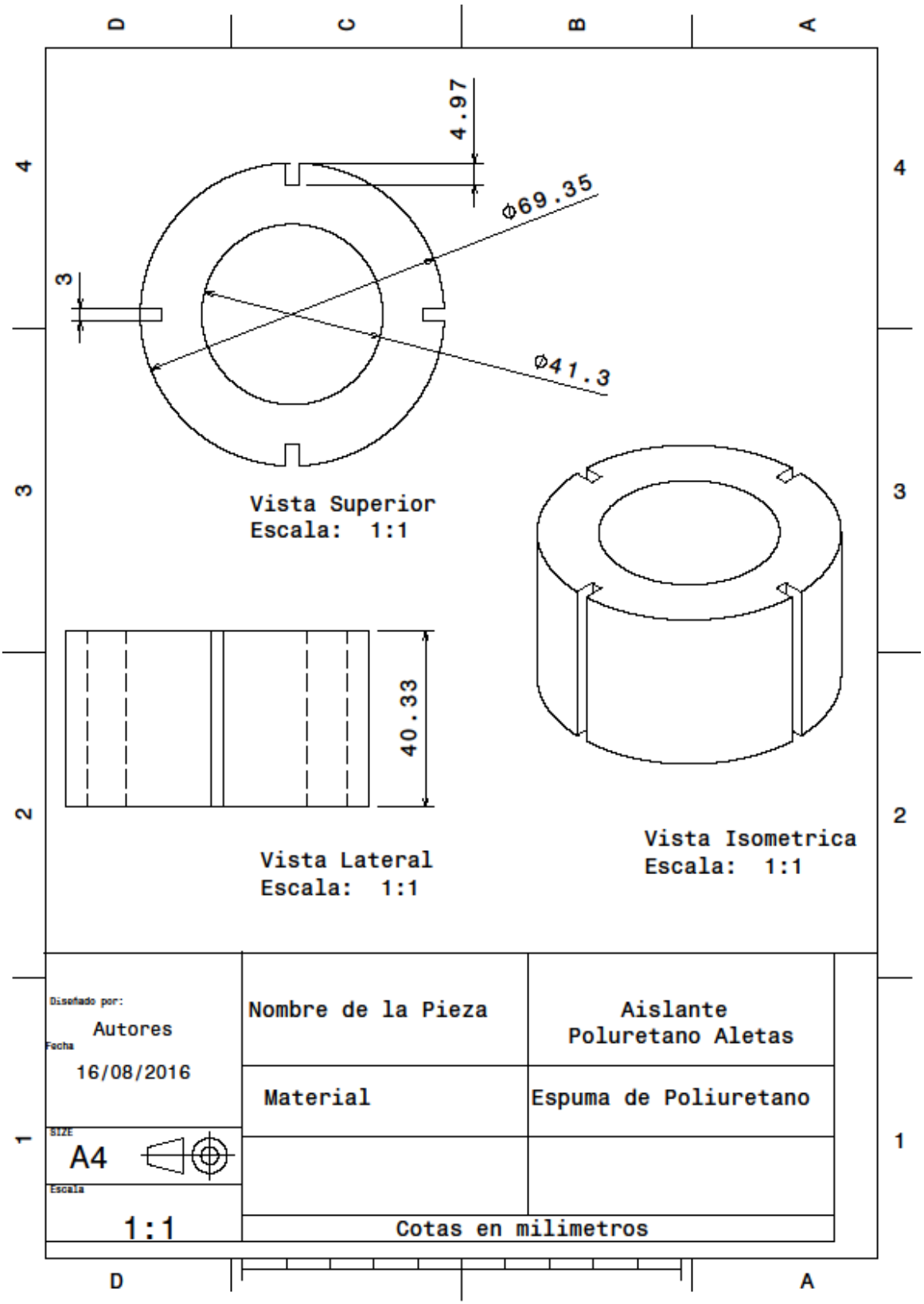
ANEXO N. TOBERA



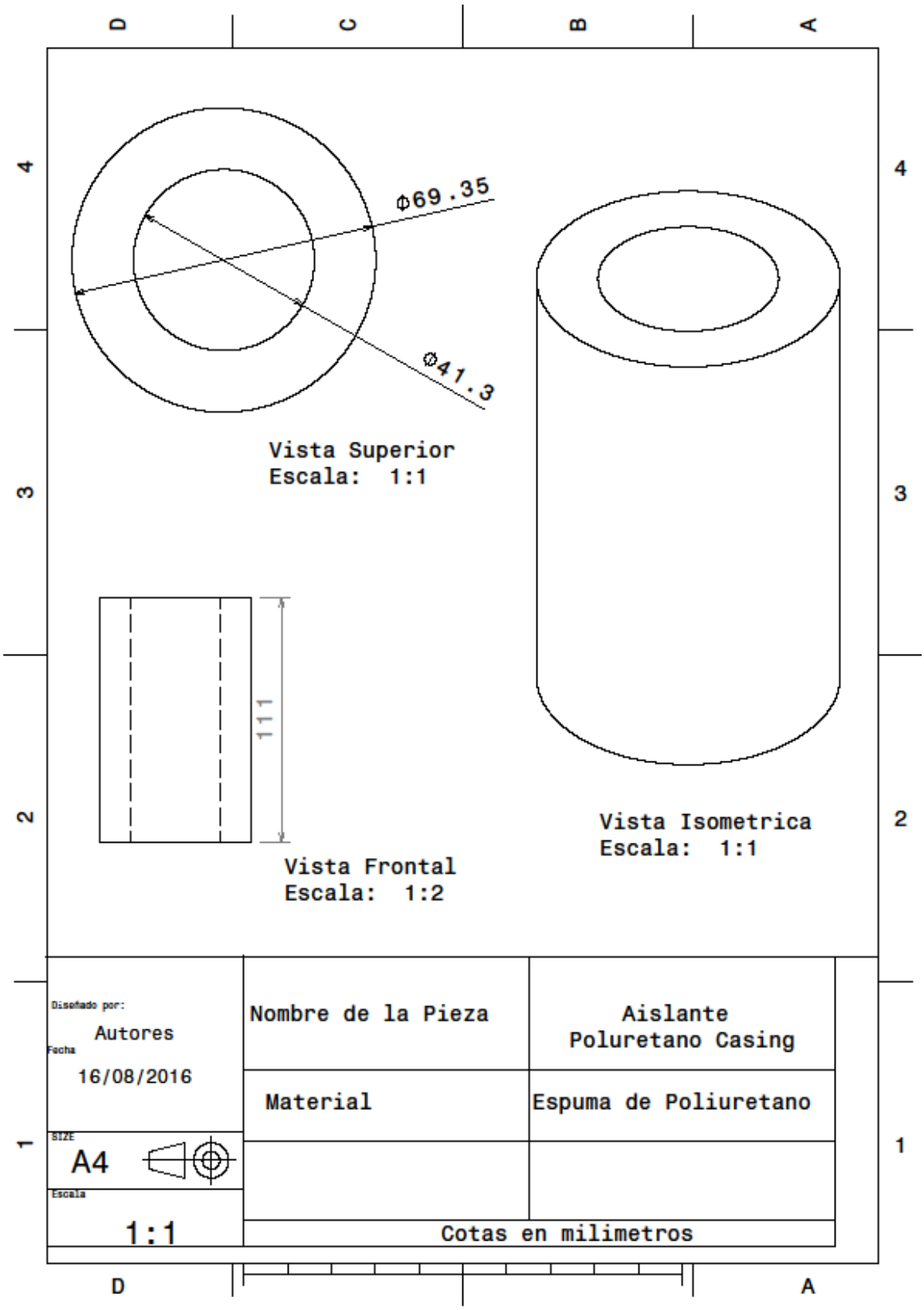
ANEXO O. PROPELENTE



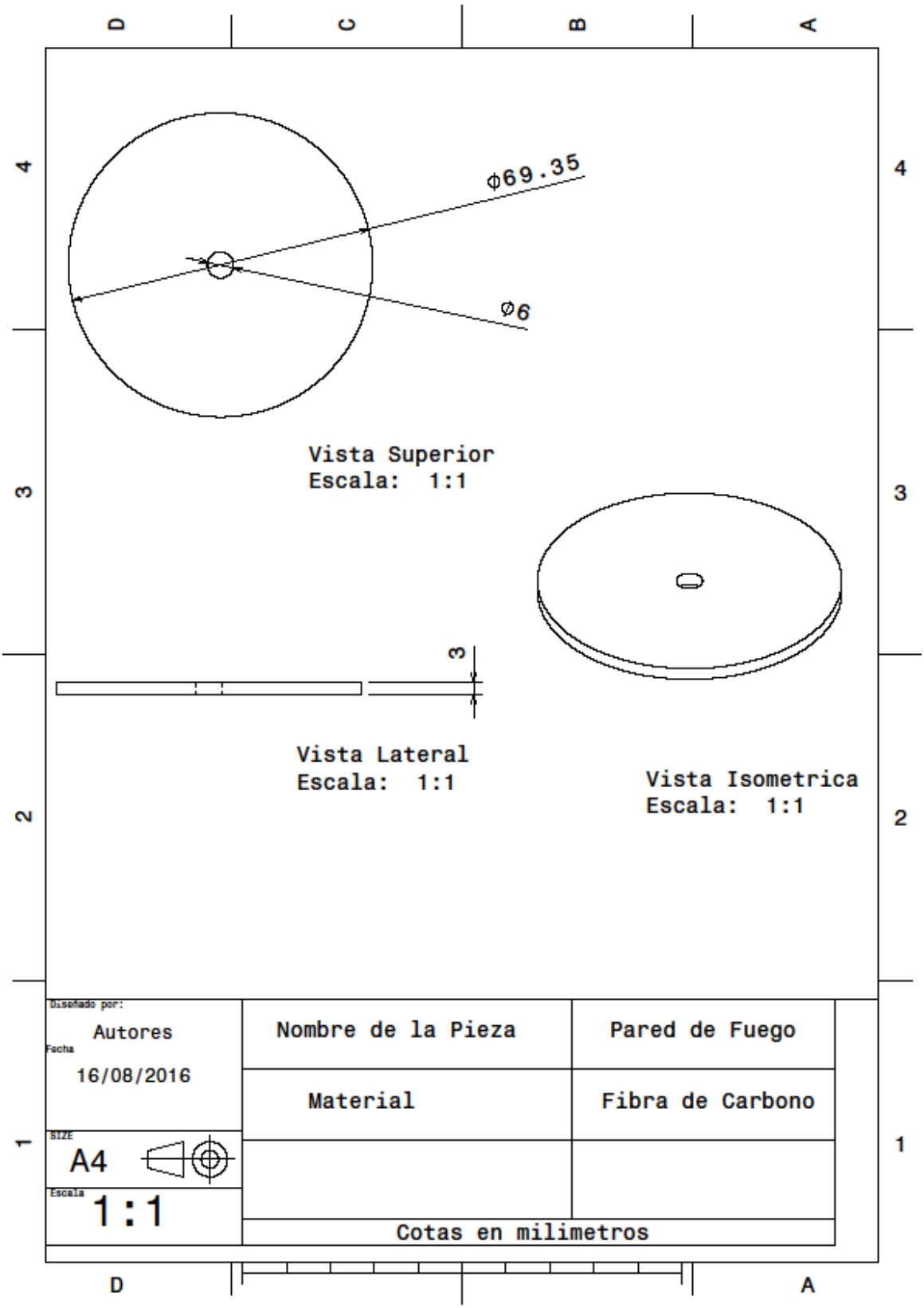
ANEXO P. AISLANTE POLIURETANO DE LAS ALETAS



ANEXO Q. AISLANTE POLIURETANO CÁMARA DE COMBUSTIÓN



ANEXO R. PARED DE FUEGO



ANEXO S. LIBRERÍA BMP 180

```
#ifndef SFE_BMP180_h
#define SFE_BMP180_h

#if defined(ARDUINO) && ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif

class SFE_BMP180
{
     public:
          SFE_BMP180(); // base type

          char begin();
               // call pressure.begin() to initialize BMP180
before use
               // returns 1 if success, 0 if failure (bad
component or I2C bus shorted?)

          char startTemperature(void);
               // command BMP180 to start a temperature
measurement
               // returns (number of ms to wait) for success, 0
for fail

          char getTemperature(double &T);
               // return temperature measurement from previous
startTemperature command
               // places returned value in T variable (deg C)
               // returns 1 for success, 0 for fail

          char startPressure(char oversampling);
               // command BMP180 to start a pressure measurement
               // oversampling: 0 - 3 for oversampling value
               // returns (number of ms to wait) for success, 0
for fail

          char getPressure(double &P, double &T);
               // return absolute pressure measurement from
previous startPressure command
               // note: requires previous temperature measurement
in variable T
               // places returned value in P variable (mbar)
               // returns 1 for success, 0 for fail

          double sealevel(double P, double A);
```

```
                // convert absolute pressure to sea-level pressure
(as used in weather data)
                // P: absolute pressure (mbar)
                // A: current altitude (meters)
                // returns sealevel pressure in mbar

          double altitude(double P, double P0);
                // convert absolute pressure to altitude (given
baseline pressure; sea-level, runway, etc.)
                // P: absolute pressure (mbar)
                // P0: fixed baseline pressure (mbar)
                // returns signed altitude in meters

          char getError(void);
                // If any library command fails, you can retrieve
an extended
                // error code using this command. Errors are from
the wire library:
                // 0 = Success
                // 1 = Data too long to fit in transmit buffer
                // 2 = Received NACK on transmit of address
                // 3 = Received NACK on transmit of data
                // 4 = Other error

     private:

          char readInt(char address, int16_t &value);
                // read an signed int (16 bits) from a BMP180
register
                // address: BMP180 register address
                // value: external signed int for returned value
(16 bits)
                // returns 1 for success, 0 for fail, with result
in value

          char readUInt(char address, uint16_t &value);
                // read an unsigned int (16 bits) from a BMP180
register
                // address: BMP180 register address
                // value: external unsigned int for returned value
(16 bits)
                // returns 1 for success, 0 for fail, with result
in value

          char readBytes(unsigned char *values, char length);
                // read a number of bytes from a BMP180 register
                // values: array of char with register address in
first location [0]
                // length: number of bytes to read back
                // returns 1 for success, 0 for fail, with read
bytes in values[] array
```

```
          char writeBytes(unsigned char *values, char length);
                // write a number of bytes to a BMP180 register
(and consecutive subsequent registers)
                // values: array of char with register address in
first location [0]
                // length: number of bytes to write
                // returns 1 for success, 0 for fail

          int16_t AC1,AC2,AC3,VB1,VB2,MB,MC,MD;
          uint16_t AC4,AC5,AC6;
          double c5,c6,mc,md,x0,x1,x2,y0,y1,y2,p0,p1,p2;
          char _error;
};

#define BMP180_ADDR 0x77 // 7-bit address

#define    BMP180_REG_CONTROL 0xF4
#define    BMP180_REG_RESULT 0xF6

#define    BMP180_COMMAND_TEMPERATURE 0x2E
#define    BMP180_COMMAND_PRESSURE0 0x34
#define    BMP180_COMMAND_PRESSURE1 0x74
#define    BMP180_COMMAND_PRESSURE2 0xB4
#define    BMP180_COMMAND_PRESSURE3 0xF4

#endif
```

Disponible en:
https://github.com/sparkfun/BMP180_Breakout/tree/master/Libraries/Arduino.

ANEXO T. LIBRERÍA I2C

```
#ifndef _I2CDEV_H_
#define _I2CDEV_H_

// -----------------------------------------------------------------
--------------
// I2C interface implementation setting
// -----------------------------------------------------------------
--------------
#define I2CDEV_IMPLEMENTATION       I2CDEV_ARDUINO_WIRE
//#define I2CDEV_IMPLEMENTATION       I2CDEV_BUILTIN_FASTWIRE

// comment this out if you are using a non-optimal
IDE/implementation setting
// but want the compiler to shut up about it
#define I2CDEV_IMPLEMENTATION_WARNINGS

// -----------------------------------------------------------------
--------------
// I2C interface implementation options
// -----------------------------------------------------------------
--------------
#define I2CDEV_ARDUINO_WIRE         1 // Wire object from Arduino
#define I2CDEV_BUILTIN_NBWIRE       2 // Tweaked Wire object from
Gene Knight's NBWire project
                                      // ^^^ NBWire implementation
is still buggy w/some interrupts!
#define I2CDEV_BUILTIN_FASTWIRE     3 // FastWire object from
Francesco Ferrara's project
#define I2CDEV_I2CMASTER_LIBRARY    4 // I2C object from
DSSCircuits I2C-Master Library at
https://github.com/DSSCircuits/I2C-Master-Library

// -----------------------------------------------------------------
--------------
// Arduino-style "Serial.print" debug constant (uncomment to
enable)
// -----------------------------------------------------------------
--------------
//#define I2CDEV_SERIAL_DEBUG

#ifdef ARDUINO
    #if ARDUINO < 100
        #include "WProgram.h"
    #else
        #include "Arduino.h"
    #endif
```

```
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        #include <Wire.h>
    #endif
    #if I2CDEV_IMPLEMENTATION == I2CDEV_I2CMASTER_LIBRARY
#include <I2C.h>
    #endif
#endif

// 1000ms default read timeout (modify with "I2Cdev::readTimeout =
[ms];")
#define I2CDEV_DEFAULT_READ_TIMEOUT     1000

class I2Cdev {
    public:
        I2Cdev();

        static int8_t readBit(uint8_t devAddr, uint8_t regAddr,
uint8_t bitNum, uint8_t *data, uint16_t
timeout=I2Cdev::readTimeout);
        static int8_t readBitW(uint8_t devAddr, uint8_t regAddr,
uint8_t bitNum, uint16_t *data, uint16_t
timeout=I2Cdev::readTimeout);
        static int8_t readBits(uint8_t devAddr, uint8_t regAddr,
uint8_t bitStart, uint8_t length, uint8_t *data, uint16_t
timeout=I2Cdev::readTimeout);
        static int8_t readBitsW(uint8_t devAddr, uint8_t regAddr,
uint8_t bitStart, uint8_t length, uint16_t *data, uint16_t
timeout=I2Cdev::readTimeout);
        static int8_t readByte(uint8_t devAddr, uint8_t regAddr,
uint8_t *data, uint16_t timeout=I2Cdev::readTimeout);
        static int8_t readWord(uint8_t devAddr, uint8_t regAddr,
uint16_t *data, uint16_t timeout=I2Cdev::readTimeout);
        static int8_t readBytes(uint8_t devAddr, uint8_t regAddr,
uint8_t length, uint8_t *data, uint16_t
timeout=I2Cdev::readTimeout);
        static int8_t readWords(uint8_t devAddr, uint8_t regAddr,
uint8_t length, uint16_t *data, uint16_t
timeout=I2Cdev::readTimeout);

        static bool writeBit(uint8_t devAddr, uint8_t regAddr,
uint8_t bitNum, uint8_t data);
        static bool writeBitW(uint8_t devAddr, uint8_t regAddr,
uint8_t bitNum, uint16_t data);
        static bool writeBits(uint8_t devAddr, uint8_t regAddr,
uint8_t bitStart, uint8_t length, uint8_t data);
        static bool writeBitsW(uint8_t devAddr, uint8_t regAddr,
uint8_t bitStart, uint8_t length, uint16_t data);
        static bool writeByte(uint8_t devAddr, uint8_t regAddr,
uint8_t data);
        static bool writeWord(uint8_t devAddr, uint8_t regAddr,
uint16_t data);
```

```
        static bool writeBytes(uint8_t devAddr, uint8_t regAddr,
uint8_t length, uint8_t *data);
        static bool writeWords(uint8_t devAddr, uint8_t regAddr,
uint8_t length, uint16_t *data);

        static uint16_t readTimeout;
};

#if I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
    ////////////////////
    // FastWire 0.24
    // This is a library to help faster programs to read I2C
devices.
    // Copyright(C) 2012
    // Francesco Ferrara
    ////////////////////

    /* Master */
    #define TW_START                0x08
    #define TW_REP_START            0x10

    /* Master Transmitter */
    #define TW_MT_SLA_ACK           0x18
    #define TW_MT_SLA_NACK          0x20
    #define TW_MT_DATA_ACK          0x28
    #define TW_MT_DATA_NACK         0x30
    #define TW_MT_ARB_LOST          0x38

    /* Master Receiver */
    #define TW_MR_ARB_LOST          0x38
    #define TW_MR_SLA_ACK           0x40
    #define TW_MR_SLA_NACK          0x48
    #define TW_MR_DATA_ACK          0x50
    #define TW_MR_DATA_NACK         0x58

    #define TW_OK                   0
    #define TW_ERROR                1

    class Fastwire {
        private:
            static boolean waitInt();

        public:
            static void setup(int khz, boolean pullup);
            static byte beginTransmission(byte device);
            static byte write(byte value);
            static byte writeBuf(byte device, byte address, byte
*data, byte num);
            static byte readBuf(byte device, byte address, byte
*data, byte num);
            static void reset();
```

```
            static byte stop();
    };
#endif

#if I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_NBWIRE
    // NBWire implementation based heavily on code by Gene Knight
<Gene@Telobot.com>
    // Originally posted on the Arduino forum at
http://arduino.cc/forum/index.php/topic,70705.0.html
    // Originally offered to the i2cdevlib project at
http://arduino.cc/forum/index.php/topic,68210.30.html

#define NBWIRE_BUFFER_LENGTH 32

class TwoWire {
        private:
            static uint8_t rxBuffer[];
            static uint8_t rxBufferIndex;
            static uint8_t rxBufferLength;

            static uint8_t txAddress;
            static uint8_t txBuffer[];
            static uint8_t txBufferIndex;
            static uint8_t txBufferLength;

            // static uint8_t transmitting;
            static void (*user_onRequest)(void);
            static void (*user_onReceive)(int);
            static void onRequestService(void);
            static void onReceiveService(uint8_t*, int);

        public:
            TwoWire();
            void begin();
            void begin(uint8_t);
            void begin(int);
            void beginTransmission(uint8_t);
            //void beginTransmission(int);
            uint8_t endTransmission(uint16_t timeout=0);
            void nbendTransmission(void (*function)(int)) ;
            uint8_t requestFrom(uint8_t, int, uint16_t timeout=0);
            //uint8_t requestFrom(int, int);
            void nbrequestFrom(uint8_t, int, void
(*function)(int));
            void send(uint8_t);
            void send(uint8_t*, uint8_t);
            //void send(int);
            void send(char*);
            uint8_t available(void);
            uint8_t receive(void);
            void onReceive(void (*)(int));
```

```
        void onRequest(void (*)(void));
};

#define TWI_READY   0
#define TWI_MRX     1
#define TWI_MTX     2
#define TWI_SRX     3
#define TWI_STX     4

#define TW_WRITE    0
#define TW_READ     1

#define TW_MT_SLA_NACK      0x20
#define TW_MT_DATA_NACK     0x30

#define CPU_FREQ            16000000L
#define TWI_FREQ            100000L
#define TWI_BUFFER_LENGTH   32

/* TWI Status is in TWSR, in the top 5 bits: TWS7 - TWS3 */

#define TW_STATUS_MASK
(_BV(TWS7)|_BV(TWS6)|_BV(TWS5)|_BV(TWS4)|_BV(TWS3))
#define TW_STATUS                   (TWSR & TW_STATUS_MASK)
#define TW_START                    0x08
#define TW_REP_START                0x10
#define TW_MT_SLA_ACK               0x18
#define TW_MT_SLA_NACK              0x20
#define TW_MT_DATA_ACK              0x28
#define TW_MT_DATA_NACK             0x30
#define TW_MT_ARB_LOST              0x38
#define TW_MR_ARB_LOST              0x38
#define TW_MR_SLA_ACK               0x40
#define TW_MR_SLA_NACK              0x48
#define TW_MR_DATA_ACK              0x50
#define TW_MR_DATA_NACK             0x58
#define TW_ST_SLA_ACK               0xA8
#define TW_ST_ARB_LOST_SLA_ACK      0xB0
#define TW_ST_DATA_ACK              0xB8
#define TW_ST_DATA_NACK             0xC0
#define TW_ST_LAST_DATA             0xC8
#define TW_SR_SLA_ACK               0x60
#define TW_SR_ARB_LOST_SLA_ACK      0x68
#define TW_SR_GCALL_ACK             0x70
#define TW_SR_ARB_LOST_GCALL_ACK    0x78
#define TW_SR_DATA_ACK              0x80
#define TW_SR_DATA_NACK             0x88
#define TW_SR_GCALL_DATA_ACK        0x90
#define TW_SR_GCALL_DATA_NACK       0x98
#define TW_SR_STOP                  0xA0
#define TW_NO_INFO                  0xF8
```

```
#define TW_BUS_ERROR                0x00

//#define _MMIO_BYTE(mem_addr) (*(volatile uint8_t *)(mem_addr))
    //#define _SFR_BYTE(sfr) _MMIO_BYTE(_SFR_ADDR(sfr))

    #ifndef sbi // set bit
        #define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
    #endif // sbi

    #ifndef cbi // clear bit
        #define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
    #endif // cbi

    extern TwoWire Wire;

#endif // I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_NBWIRE

#endif /* _I2CDEV_H_ */
```

Disponible en: https://github.com/jrowberg/i2cdevlib/tree/master/Arduino.

ANEXO U. LIBRERÍA WIRE

```
#ifndef TwoWire_h
#define TwoWire_h
#include <inttypes.h>
#include "Stream.h"
#define BUFFER_LENGTH 32
// WIRE_HAS_END means Wire has end()
#define WIRE_HAS_END 1

class TwoWire : public Stream
{
  private:
    static uint8_t rxBuffer[];
    static uint8_t rxBufferIndex;
    static uint8_t rxBufferLength;

    static uint8_t txAddress;
    static uint8_t txBuffer[];
    static uint8_t txBufferIndex;
    static uint8_t txBufferLength;

    static uint8_t transmitting;
    static void (*user_onRequest)(void);
    static void (*user_onReceive)(int);
    static void onRequestService(void);
    static void onReceiveService(uint8_t*, int);
  public:
    TwoWire();
    void begin();
    void begin(uint8_t);
    void begin(int);
    void end();
    void setClock(uint32_t);
    void beginTransmission(uint8_t);
    void beginTransmission(int);
    uint8_t endTransmission(void);
    uint8_t endTransmission(uint8_t);
    uint8_t requestFrom(uint8_t, uint8_t);
    uint8_t requestFrom(uint8_t, uint8_t, uint8_t);
     uint8_t requestFrom(uint8_t, uint8_t, uint32_t, uint8_t,
uint8_t);
    uint8_t requestFrom(int, int);
    uint8_t requestFrom(int, int, int);
    virtual size_t write(uint8_t);
    virtual size_t write(const uint8_t *, size_t);
    virtual int available(void);
    virtual int read(void);
    virtual int peek(void);
    virtual void flush(void);
```

```
    void onReceive( void (*)(int) );
    void onRequest( void (*)(void) );

    inline size_t write(unsigned long n) { return
write((uint8_t)n); }
    inline size_t write(long n) { return write((uint8_t)n); }
    inline size_t write(unsigned int n) { return
write((uint8_t)n); }
    inline size_t write(int n) { return write((uint8_t)n); }
    using Print::write;
};

extern TwoWire Wire;

#endif
```

Disponible en: Esta librería se encuentra dentro de las librerías por defecto que incorpora el software ARDUINO.

ANEXO V. LIBRERÍA SD

```
#ifndef __SD_H__
#define __SD_H__

#include <Arduino.h>

#include <utility/SdFat.h>
#include <utility/SdFatUtil.h>

#define FILE_READ O_READ
#define FILE_WRITE (O_READ | O_WRITE | O_CREAT)

namespace SDLib {

class File : public Stream {
 private:
  char _name[13]; // our name
  SdFile *_file;  // underlying file pointer

public:
  File(SdFile f, const char *name);     // wraps an underlying
SdFile
  File(void);      // 'empty' constructor
  virtual size_t write(uint8_t);
  virtual size_t write(const uint8_t *buf, size_t size);
  virtual int read();
  virtual int peek();
  virtual int available();
  virtual void flush();
  int read(void *buf, uint16_t nbyte);
  boolean seek(uint32_t pos);
  uint32_t position();
  uint32_t size();
  void close();
  operator bool();
  char * name();

  boolean isDirectory(void);
  File openNextFile(uint8_t mode = O_RDONLY);
  void rewindDirectory(void);

  using Print::write;
};

class SDClass {

private:
  // These are required for initialisation and use of sdfatlib
```

```
  Sd2Card card;
  SdVolume volume;
  SdFile root;

  // my quick&dirty iterator, should be replaced
  SdFile getParentDir(const char *filepath, int *indx);
public:
  // This needs to be called to set up the connection to the SD
card
  // before other methods are used.
  boolean begin(uint8_t csPin = SD_CHIP_SELECT_PIN);

  // Open the specified file/directory with the supplied mode
(e.g. read or
  // write, etc). Returns a File object for interacting with the
file.
  // Note that currently only one file can be open at a time.
  File open(const char *filename, uint8_t mode = FILE_READ);
  File open(const String &filename, uint8_t mode = FILE_READ) {
return open( filename.c_str(), mode ); }

  // Methods to determine if the requested file path exists.
  boolean exists(const char *filepath);
  boolean exists(const String &filepath) { return
exists(filepath.c_str()); }

  // Create the requested directory heirarchy--if intermediate
directories
  // do not exist they will be created.
  boolean mkdir(const char *filepath);
  boolean mkdir(const String &filepath) { return
mkdir(filepath.c_str()); }

// Delete the file.
boolean remove(const char *filepath);
  boolean remove(const String &filepath) { return
remove(filepath.c_str()); }

  boolean rmdir(const char *filepath);
  boolean rmdir(const String &filepath) { return
rmdir(filepath.c_str()); }

private:

  // This is used to determine the mode used to open a file
  // it's here because it's the easiest place to pass the
  // information through the directory walking function. But
// it's probably not the best place for it.
  // It shouldn't be set directly--it is set via the parameters to
`open`.
  int fileOpenMode;
```

```
  friend class File;
  friend boolean callback_openPath(SdFile&, const char *, boolean,
void *);
};

extern SDClass SD;

};

// We enclose File and SD classes in namespace SDLib to avoid
conflicts
// with others legacy libraries that redefines File class.

// This ensure compatibility with sketches that uses only SD
library
using namespace SDLib;

// This allows sketches to use SDLib::File with other libraries
(in the
// sketch you must use SDFile instead of File to disambiguate)
typedef SDLib::File    SDFile;
typedef SDLib::SDClass SDFileSystemClass;
#define SDFileSystem   SDLib::SD

#endif
```

Disponible en: Esta librería se encuentra dentro de las librerías por defecto que incorpora el software ARDUINO.

ANEXO W. LIBRERÍA MPU6050

```
#ifndef _MPU6050_H_
#define _MPU6050_H_

#include "I2Cdev.h"

// supporting link:
http://forum.arduino.cc/index.php?&topic=143444.msg1079517#msg1079
517
// also:
http://forum.arduino.cc/index.php?&topic=141571.msg1062899#msg1062
899s
#ifndef __arm__
#include <avr/pgmspace.h>
#else
#define PROGMEM /* empty */
#define pgm_read_byte(x) (*(x))
#define pgm_read_word(x) (*(x))
#define pgm_read_float(x) (*(x))
#define PSTR(STR) STR
#endif


#define MPU6050_ADDRESS_AD0_LOW     0x68 // address pin low (GND),
default for InvenSense evaluation board
#define MPU6050_ADDRESS_AD0_HIGH    0x69 // address pin high (VCC)
#define MPU6050_DEFAULT_ADDRESS     MPU6050_ADDRESS_AD0_LOW

#define MPU6050_RA_XG_OFFS_TC       0x00 //[7] PWR_MODE, [6:1]
XG_OFFS_TC, [0] OTP_BNK_VLD
#define MPU6050_RA_YG_OFFS_TC       0x01 //[7] PWR_MODE, [6:1]
YG_OFFS_TC, [0] OTP_BNK_VLD
#define MPU6050_RA_ZG_OFFS_TC       0x02 //[7] PWR_MODE, [6:1]
ZG_OFFS_TC, [0] OTP_BNK_VLD
#define MPU6050_RA_X_FINE_GAIN      0x03 //[7:0] X_FINE_GAIN
#define MPU6050_RA_Y_FINE_GAIN      0x04 //[7:0] Y_FINE_GAIN
#define MPU6050_RA_Z_FINE_GAIN      0x05 //[7:0] Z_FINE_GAIN
#define MPU6050_RA_XA_OFFS_H        0x06 //[15:0] XA_OFFS
#define MPU6050_RA_XA_OFFS_L_TC     0x07
#define MPU6050_RA_YA_OFFS_H        0x08 //[15:0] YA_OFFS
#define MPU6050_RA_YA_OFFS_L_TC     0x09
#define MPU6050_RA_ZA_OFFS_H        0x0A //[15:0] ZA_OFFS
#define MPU6050_RA_ZA_OFFS_L_TC     0x0B
#define MPU6050_RA_XG_OFFS_USRH     0x13 //[15:0] XG_OFFS_USR
#define MPU6050_RA_XG_OFFS_USRL     0x14
#define MPU6050_RA_YG_OFFS_USRH     0x15 //[15:0] YG_OFFS_USR
#define MPU6050_RA_YG_OFFS_USRL     0x16
```

```
#define MPU6050_RA_ZG_OFFS_USRH     0x17 //[15:0] ZG_OFFS_USR
#define MPU6050_RA_ZG_OFFS_USRL     0x18
#define MPU6050_RA_SMPLRT_DIV       0x19
#define MPU6050_RA_CONFIG           0x1A
#define MPU6050_RA_GYRO_CONFIG      0x1B
#define MPU6050_RA_ACCEL_CONFIG     0x1C
#define MPU6050_RA_FF_THR           0x1D
#define MPU6050_RA_FF_DUR           0x1E
#define MPU6050_RA_MOT_THR          0x1F
#define MPU6050_RA_MOT_DUR          0x20
#define MPU6050_RA_ZRMOT_THR        0x21
#define MPU6050_RA_ZRMOT_DUR        0x22
#define MPU6050_RA_FIFO_EN          0x23
#define MPU6050_RA_I2C_MST_CTRL     0x24
#define MPU6050_RA_I2C_SLV0_ADDR    0x25
#define MPU6050_RA_I2C_SLV0_REG     0x26
#define MPU6050_RA_I2C_SLV0_CTRL    0x27
#define MPU6050_RA_I2C_SLV1_ADDR    0x28
#define MPU6050_RA_I2C_SLV1_REG     0x29
#define MPU6050_RA_I2C_SLV1_CTRL    0x2A
#define MPU6050_RA_I2C_SLV2_ADDR    0x2B
#define MPU6050_RA_I2C_SLV2_REG     0x2C
#define MPU6050_RA_I2C_SLV2_CTRL    0x2D
#define MPU6050_RA_I2C_SLV3_ADDR    0x2E
#define MPU6050_RA_I2C_SLV3_REG     0x2F
#define MPU6050_RA_I2C_SLV3_CTRL    0x30
#define MPU6050_RA_I2C_SLV4_ADDR    0x31
#define MPU6050_RA_I2C_SLV4_REG     0x32
#define MPU6050_RA_I2C_SLV4_DO      0x33
#define MPU6050_RA_I2C_SLV4_CTRL    0x34
#define MPU6050_RA_I2C_SLV4_DI      0x35
#define MPU6050_RA_I2C_MST_STATUS   0x36
#define MPU6050_RA_INT_PIN_CFG      0x37
#define MPU6050_RA_INT_ENABLE       0x38
#define MPU6050_RA_DMP_INT_STATUS   0x39
#define MPU6050_RA_INT_STATUS       0x3A
#define MPU6050_RA_ACCEL_XOUT_H     0x3B
#define MPU6050_RA_ACCEL_XOUT_L     0x3C
#define MPU6050_RA_ACCEL_YOUT_H     0x3D
#define MPU6050_RA_ACCEL_YOUT_L     0x3E
#define MPU6050_RA_ACCEL_ZOUT_H     0x3F
#define MPU6050_RA_ACCEL_ZOUT_L     0x40
#define MPU6050_RA_TEMP_OUT_H       0x41
#define MPU6050_RA_TEMP_OUT_L       0x42
#define MPU6050_RA_GYRO_XOUT_H      0x43
#define MPU6050_RA_GYRO_XOUT_L      0x44
#define MPU6050_RA_GYRO_YOUT_H      0x45
#define MPU6050_RA_GYRO_YOUT_L      0x46
#define MPU6050_RA_GYRO_ZOUT_H      0x47
#define MPU6050_RA_GYRO_ZOUT_L      0x48
#define MPU6050_RA_EXT_SENS_DATA_00 0x49
```

```
#define MPU6050_RA_EXT_SENS_DATA_01 0x4A
#define MPU6050_RA_EXT_SENS_DATA_02 0x4B
#define MPU6050_RA_EXT_SENS_DATA_03 0x4C
#define MPU6050_RA_EXT_SENS_DATA_04 0x4D
#define MPU6050_RA_EXT_SENS_DATA_05 0x4E
#define MPU6050_RA_EXT_SENS_DATA_06 0x4F
#define MPU6050_RA_EXT_SENS_DATA_07 0x50
#define MPU6050_RA_EXT_SENS_DATA_08 0x51
#define MPU6050_RA_EXT_SENS_DATA_09 0x52
#define MPU6050_RA_EXT_SENS_DATA_10 0x53
#define MPU6050_RA_EXT_SENS_DATA_11 0x54
#define MPU6050_RA_EXT_SENS_DATA_12 0x55
#define MPU6050_RA_EXT_SENS_DATA_13 0x56
#define MPU6050_RA_EXT_SENS_DATA_14 0x57
#define MPU6050_RA_EXT_SENS_DATA_15 0x58
#define MPU6050_RA_EXT_SENS_DATA_16 0x59
#define MPU6050_RA_EXT_SENS_DATA_17 0x5A
#define MPU6050_RA_EXT_SENS_DATA_18 0x5B
#define MPU6050_RA_EXT_SENS_DATA_19 0x5C
#define MPU6050_RA_EXT_SENS_DATA_20 0x5D
#define MPU6050_RA_EXT_SENS_DATA_21 0x5E
#define MPU6050_RA_EXT_SENS_DATA_22 0x5F
#define MPU6050_RA_EXT_SENS_DATA_23 0x60
#define MPU6050_RA_MOT_DETECT_STATUS    0x61
#define MPU6050_RA_I2C_SLV0_DO      0x63
#define MPU6050_RA_I2C_SLV1_DO      0x64
#define MPU6050_RA_I2C_SLV2_DO      0x65
#define MPU6050_RA_I2C_SLV3_DO      0x66
#define MPU6050_RA_I2C_MST_DELAY_CTRL   0x67
#define MPU6050_RA_SIGNAL_PATH_RESET    0x68
#define MPU6050_RA_MOT_DETECT_CTRL      0x69
#define MPU6050_RA_USER_CTRL        0x6A
#define MPU6050_RA_PWR_MGMT_1       0x6B
#define MPU6050_RA_PWR_MGMT_2       0x6C
#define MPU6050_RA_BANK_SEL         0x6D
#define MPU6050_RA_MEM_START_ADDR   0x6E
#define MPU6050_RA_MEM_R_W          0x6F
#define MPU6050_RA_DMP_CFG_1        0x70
#define MPU6050_RA_DMP_CFG_2        0x71
#define MPU6050_RA_FIFO_COUNTH      0x72
#define MPU6050_RA_FIFO_COUNTL      0x73
#define MPU6050_RA_FIFO_R_W         0x74
#define MPU6050_RA_WHO_AM_I         0x75

#define MPU6050_TC_PWR_MODE_BIT     7
#define MPU6050_TC_OFFSET_BIT       6
#define MPU6050_TC_OFFSET_LENGTH    6
#define MPU6050_TC_OTP_BNK_VLD_BIT  0

#define MPU6050_VDDIO_LEVEL_VLOGIC  0
#define MPU6050_VDDIO_LEVEL_VDD     1
```

```
#define MPU6050_CFG_EXT_SYNC_SET_BIT    5
#define MPU6050_CFG_EXT_SYNC_SET_LENGTH 3
#define MPU6050_CFG_DLPF_CFG_BIT    2
#define MPU6050_CFG_DLPF_CFG_LENGTH 3

#define MPU6050_EXT_SYNC_DISABLED       0x0
#define MPU6050_EXT_SYNC_TEMP_OUT_L     0x1
#define MPU6050_EXT_SYNC_GYRO_XOUT_L    0x2
#define MPU6050_EXT_SYNC_GYRO_YOUT_L    0x3
#define MPU6050_EXT_SYNC_GYRO_ZOUT_L    0x4
#define MPU6050_EXT_SYNC_ACCEL_XOUT_L   0x5
#define MPU6050_EXT_SYNC_ACCEL_YOUT_L   0x6
#define MPU6050_EXT_SYNC_ACCEL_ZOUT_L   0x7

#define MPU6050_DLPF_BW_256         0x00
#define MPU6050_DLPF_BW_188         0x01
#define MPU6050_DLPF_BW_98          0x02
#define MPU6050_DLPF_BW_42          0x03
#define MPU6050_DLPF_BW_20          0x04
#define MPU6050_DLPF_BW_10          0x05
#define MPU6050_DLPF_BW_5           0x06

#define MPU6050_GCONFIG_FS_SEL_BIT      4
#define MPU6050_GCONFIG_FS_SEL_LENGTH   2

#define MPU6050_GYRO_FS_250         0x00
#define MPU6050_GYRO_FS_500         0x01
#define MPU6050_GYRO_FS_1000        0x02
#define MPU6050_GYRO_FS_2000        0x03

#define MPU6050_ACONFIG_XA_ST_BIT           7
#define MPU6050_ACONFIG_YA_ST_BIT           6
#define MPU6050_ACONFIG_ZA_ST_BIT           5
#define MPU6050_ACONFIG_AFS_SEL_BIT         4
#define MPU6050_ACONFIG_AFS_SEL_LENGTH      2
#define MPU6050_ACONFIG_ACCEL_HPF_BIT       2
#define MPU6050_ACONFIG_ACCEL_HPF_LENGTH    3

#define MPU6050_ACCEL_FS_2          0x00
#define MPU6050_ACCEL_FS_4          0x01
#define MPU6050_ACCEL_FS_8          0x02
#define MPU6050_ACCEL_FS_16         0x03

#define MPU6050_DHPF_RESET          0x00
#define MPU6050_DHPF_5              0x01
#define MPU6050_DHPF_2P5            0x02
#define MPU6050_DHPF_1P25           0x03
#define MPU6050_DHPF_0P63           0x04
#define MPU6050_DHPF_HOLD           0x07
```

```
#define MPU6050_TEMP_FIFO_EN_BIT    7
#define MPU6050_XG_FIFO_EN_BIT      6
#define MPU6050_YG_FIFO_EN_BIT      5
#define MPU6050_ZG_FIFO_EN_BIT      4
#define MPU6050_ACCEL_FIFO_EN_BIT   3
#define MPU6050_SLV2_FIFO_EN_BIT    2
#define MPU6050_SLV1_FIFO_EN_BIT    1
#define MPU6050_SLV0_FIFO_EN_BIT    0

#define MPU6050_MULT_MST_EN_BIT     7
#define MPU6050_WAIT_FOR_ES_BIT     6
#define MPU6050_SLV_3_FIFO_EN_BIT   5
#define MPU6050_I2C_MST_P_NSR_BIT   4
#define MPU6050_I2C_MST_CLK_BIT     3
#define MPU6050_I2C_MST_CLK_LENGTH  4

#define MPU6050_CLOCK_DIV_348       0x0
#define MPU6050_CLOCK_DIV_333       0x1
#define MPU6050_CLOCK_DIV_320       0x2
#define MPU6050_CLOCK_DIV_308       0x3
#define MPU6050_CLOCK_DIV_296       0x4
#define MPU6050_CLOCK_DIV_286       0x5
#define MPU6050_CLOCK_DIV_276       0x6
#define MPU6050_CLOCK_DIV_267       0x7
#define MPU6050_CLOCK_DIV_258       0x8
#define MPU6050_CLOCK_DIV_500       0x9
#define MPU6050_CLOCK_DIV_471       0xA
#define MPU6050_CLOCK_DIV_444       0xB
#define MPU6050_CLOCK_DIV_421       0xC
#define MPU6050_CLOCK_DIV_400       0xD
#define MPU6050_CLOCK_DIV_381       0xE
#define MPU6050_CLOCK_DIV_364       0xF

#define MPU6050_I2C_SLV_RW_BIT      7
#define MPU6050_I2C_SLV_ADDR_BIT    6
#define MPU6050_I2C_SLV_ADDR_LENGTH 7
#define MPU6050_I2C_SLV_EN_BIT      7
#define MPU6050_I2C_SLV_BYTE_SW_BIT 6
#define MPU6050_I2C_SLV_REG_DIS_BIT 5
#define MPU6050_I2C_SLV_GRP_BIT     4
#define MPU6050_I2C_SLV_LEN_BIT     3
#define MPU6050_I2C_SLV_LEN_LENGTH  4

#define MPU6050_I2C_SLV4_RW_BIT         7
#define MPU6050_I2C_SLV4_ADDR_BIT       6
#define MPU6050_I2C_SLV4_ADDR_LENGTH    7
#define MPU6050_I2C_SLV4_EN_BIT         7
#define MPU6050_I2C_SLV4_INT_EN_BIT     6
#define MPU6050_I2C_SLV4_REG_DIS_BIT    5
#define MPU6050_I2C_SLV4_MST_DLY_BIT    4
#define MPU6050_I2C_SLV4_MST_DLY_LENGTH 5
```

```
#define MPU6050_MST_PASS_THROUGH_BIT    7
#define MPU6050_MST_I2C_SLV4_DONE_BIT   6
#define MPU6050_MST_I2C_LOST_ARB_BIT    5
#define MPU6050_MST_I2C_SLV4_NACK_BIT   4
#define MPU6050_MST_I2C_SLV3_NACK_BIT   3
#define MPU6050_MST_I2C_SLV2_NACK_BIT   2
#define MPU6050_MST_I2C_SLV1_NACK_BIT   1
#define MPU6050_MST_I2C_SLV0_NACK_BIT   0

#define MPU6050_INTCFG_INT_LEVEL_BIT        7
#define MPU6050_INTCFG_INT_OPEN_BIT         6
#define MPU6050_INTCFG_LATCH_INT_EN_BIT     5
#define MPU6050_INTCFG_INT_RD_CLEAR_BIT     4
#define MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT  3
#define MPU6050_INTCFG_FSYNC_INT_EN_BIT     2
#define MPU6050_INTCFG_I2C_BYPASS_EN_BIT    1
#define MPU6050_INTCFG_CLKOUT_EN_BIT        0

#define MPU6050_INTMODE_ACTIVEHIGH  0x00
#define MPU6050_INTMODE_ACTIVELOW   0x01

#define MPU6050_INTDRV_PUSHPULL     0x00
#define MPU6050_INTDRV_OPENDRAIN    0x01

#define MPU6050_INTLATCH_50USPULSE  0x00
#define MPU6050_INTLATCH_WAITCLEAR  0x01

#define MPU6050_INTCLEAR_STATUSREAD 0x00
#define MPU6050_INTCLEAR_ANYREAD    0x01

#define MPU6050_INTERRUPT_FF_BIT            7
#define MPU6050_INTERRUPT_MOT_BIT           6
#define MPU6050_INTERRUPT_ZMOT_BIT          5
#define MPU6050_INTERRUPT_FIFO_OFLOW_BIT    4
#define MPU6050_INTERRUPT_I2C_MST_INT_BIT   3
#define MPU6050_INTERRUPT_PLL_RDY_INT_BIT   2
#define MPU6050_INTERRUPT_DMP_INT_BIT       1
#define MPU6050_INTERRUPT_DATA_RDY_BIT      0

// TODO: figure out what these actually do
// UMPL source code is not very obivous
#define MPU6050_DMPINT_5_BIT            5
#define MPU6050_DMPINT_4_BIT            4
#define MPU6050_DMPINT_3_BIT            3
#define MPU6050_DMPINT_2_BIT            2
#define MPU6050_DMPINT_1_BIT            1
#define MPU6050_DMPINT_0_BIT            0

#define MPU6050_MOTION_MOT_XNEG_BIT     7
#define MPU6050_MOTION_MOT_XPOS_BIT     6
```

```
#define MPU6050_MOTION_MOT_YNEG_BIT     5
#define MPU6050_MOTION_MOT_YPOS_BIT     4
#define MPU6050_MOTION_MOT_ZNEG_BIT     3
#define MPU6050_MOTION_MOT_ZPOS_BIT     2
#define MPU6050_MOTION_MOT_ZRMOT_BIT    0

#define MPU6050_DELAYCTRL_DELAY_ES_SHADOW_BIT   7
#define MPU6050_DELAYCTRL_I2C_SLV4_DLY_EN_BIT   4
#define MPU6050_DELAYCTRL_I2C_SLV3_DLY_EN_BIT   3
#define MPU6050_DELAYCTRL_I2C_SLV2_DLY_EN_BIT   2
#define MPU6050_DELAYCTRL_I2C_SLV1_DLY_EN_BIT   1
#define MPU6050_DELAYCTRL_I2C_SLV0_DLY_EN_BIT   0

#define MPU6050_PATHRESET_GYRO_RESET_BIT    2
#define MPU6050_PATHRESET_ACCEL_RESET_BIT   1
#define MPU6050_PATHRESET_TEMP_RESET_BIT    0

#define MPU6050_DETECT_ACCEL_ON_DELAY_BIT       5
#define MPU6050_DETECT_ACCEL_ON_DELAY_LENGTH    2
#define MPU6050_DETECT_FF_COUNT_BIT             3
#define MPU6050_DETECT_FF_COUNT_LENGTH          2
#define MPU6050_DETECT_MOT_COUNT_BIT            1
#define MPU6050_DETECT_MOT_COUNT_LENGTH         2

#define MPU6050_DETECT_DECREMENT_RESET  0x0
#define MPU6050_DETECT_DECREMENT_1      0x1
#define MPU6050_DETECT_DECREMENT_2      0x2
#define MPU6050_DETECT_DECREMENT_4      0x3

#define MPU6050_USERCTRL_DMP_EN_BIT             7
#define MPU6050_USERCTRL_FIFO_EN_BIT            6
#define MPU6050_USERCTRL_I2C_MST_EN_BIT         5
#define MPU6050_USERCTRL_I2C_IF_DIS_BIT         4
#define MPU6050_USERCTRL_DMP_RESET_BIT          3
#define MPU6050_USERCTRL_FIFO_RESET_BIT         2
#define MPU6050_USERCTRL_I2C_MST_RESET_BIT      1
#define MPU6050_USERCTRL_SIG_COND_RESET_BIT     0

#define MPU6050_PWR1_DEVICE_RESET_BIT   7
#define MPU6050_PWR1_SLEEP_BIT          6
#define MPU6050_PWR1_CYCLE_BIT          5
#define MPU6050_PWR1_TEMP_DIS_BIT       3
#define MPU6050_PWR1_CLKSEL_BIT         2
#define MPU6050_PWR1_CLKSEL_LENGTH      3

#define MPU6050_CLOCK_INTERNAL          0x00
#define MPU6050_CLOCK_PLL_XGYRO         0x01
#define MPU6050_CLOCK_PLL_YGYRO         0x02
#define MPU6050_CLOCK_PLL_ZGYRO         0x03
#define MPU6050_CLOCK_PLL_EXT32K        0x04
#define MPU6050_CLOCK_PLL_EXT19M        0x05
```

```
#define MPU6050_CLOCK_KEEP_RESET        0x07

#define MPU6050_PWR2_LP_WAKE_CTRL_BIT       7
#define MPU6050_PWR2_LP_WAKE_CTRL_LENGTH    2
#define MPU6050_PWR2_STBY_XA_BIT            5
#define MPU6050_PWR2_STBY_YA_BIT            4
#define MPU6050_PWR2_STBY_ZA_BIT            3
#define MPU6050_PWR2_STBY_XG_BIT            2
#define MPU6050_PWR2_STBY_YG_BIT            1
#define MPU6050_PWR2_STBY_ZG_BIT            0

#define MPU6050_WAKE_FREQ_1P25      0x0
#define MPU6050_WAKE_FREQ_2P5       0x1
#define MPU6050_WAKE_FREQ_5         0x2
#define MPU6050_WAKE_FREQ_10        0x3

#define MPU6050_BANKSEL_PRFTCH_EN_BIT       6
#define MPU6050_BANKSEL_CFG_USER_BANK_BIT   5
#define MPU6050_BANKSEL_MEM_SEL_BIT         4
#define MPU6050_BANKSEL_MEM_SEL_LENGTH      5

#define MPU6050_WHO_AM_I_BIT        6
#define MPU6050_WHO_AM_I_LENGTH     6

#define MPU6050_DMP_MEMORY_BANKS        8
#define MPU6050_DMP_MEMORY_BANK_SIZE    256
#define MPU6050_DMP_MEMORY_CHUNK_SIZE   16

// note: DMP code memory blocks defined at end of header file

class MPU6050 {
    public:
        MPU6050();
        MPU6050(uint8_t address);

        void initialize();
        bool testConnection();

        // AUX_VDDIO register
        uint8_t getAuxVDDIOLevel();
        void setAuxVDDIOLevel(uint8_t level);

// SMPLRT_DIV register
uint8_t getRate();
        void setRate(uint8_t rate);

// CONFIG register
uint8_t getExternalFrameSync();
        void setExternalFrameSync(uint8_t sync);
        uint8_t getDLPFMode();
        void setDLPFMode(uint8_t bandwidth);
```

```
// GYRO_CONFIG register
uint8_t getFullScaleGyroRange();
        void setFullScaleGyroRange(uint8_t range);

// ACCEL_CONFIG register
bool getAccelXSelfTest();
        void setAccelXSelfTest(bool enabled);
        bool getAccelYSelfTest();
        void setAccelYSelfTest(bool enabled);
        bool getAccelZSelfTest();
        void setAccelZSelfTest(bool enabled);
        uint8_t getFullScaleAccelRange();
        void setFullScaleAccelRange(uint8_t range);
        uint8_t getDHPFMode();
        void setDHPFMode(uint8_t mode);

// FF_THR register
uint8_t getFreefallDetectionThreshold();
        void setFreefallDetectionThreshold(uint8_t threshold);

// FF_DUR register
uint8_t getFreefallDetectionDuration();
        void setFreefallDetectionDuration(uint8_t duration);

// MOT_THR register
uint8_t getMotionDetectionThreshold();
        void setMotionDetectionThreshold(uint8_t threshold);

// MOT_DUR register
uint8_t getMotionDetectionDuration();
        void setMotionDetectionDuration(uint8_t duration);

// ZRMOT_THR register
uint8_t getZeroMotionDetectionThreshold();
        void setZeroMotionDetectionThreshold(uint8_t threshold);

// ZRMOT_DUR register
uint8_t getZeroMotionDetectionDuration();
        void setZeroMotionDetectionDuration(uint8_t duration);

// FIFO_EN register
bool getTempFIFOEnabled();
        void setTempFIFOEnabled(bool enabled);
        bool getXGyroFIFOEnabled();
        void setXGyroFIFOEnabled(bool enabled);
        bool getYGyroFIFOEnabled();
        void setYGyroFIFOEnabled(bool enabled);
        bool getZGyroFIFOEnabled();
        void setZGyroFIFOEnabled(bool enabled);
        bool getAccelFIFOEnabled();
```

```
        void setAccelFIFOEnabled(bool enabled);
        bool getSlave2FIFOEnabled();
        void setSlave2FIFOEnabled(bool enabled);
        bool getSlave1FIFOEnabled();
        void setSlave1FIFOEnabled(bool enabled);
        bool getSlave0FIFOEnabled();
        void setSlave0FIFOEnabled(bool enabled);

// I2C_MST_CTRL register
bool getMultiMasterEnabled();
        void setMultiMasterEnabled(bool enabled);
        bool getWaitForExternalSensorEnabled();
        void setWaitForExternalSensorEnabled(bool enabled);
        bool getSlave3FIFOEnabled();
        void setSlave3FIFOEnabled(bool enabled);
        bool getSlaveReadWriteTransitionEnabled();
        void setSlaveReadWriteTransitionEnabled(bool enabled);
        uint8_t getMasterClockSpeed();
        void setMasterClockSpeed(uint8_t speed);

// I2C_SLV* registers (Slave 0-3)
uint8_t getSlaveAddress(uint8_t num);
        void setSlaveAddress(uint8_t num, uint8_t address);
        uint8_t getSlaveRegister(uint8_t num);
        void setSlaveRegister(uint8_t num, uint8_t reg);
        bool getSlaveEnabled(uint8_t num);
        void setSlaveEnabled(uint8_t num, bool enabled);
        bool getSlaveWordByteSwap(uint8_t num);
        void setSlaveWordByteSwap(uint8_t num, bool enabled);
        bool getSlaveWriteMode(uint8_t num);
        void setSlaveWriteMode(uint8_t num, bool mode);
        bool getSlaveWordGroupOffset(uint8_t num);
        void setSlaveWordGroupOffset(uint8_t num, bool enabled);
        uint8_t getSlaveDataLength(uint8_t num);
        void setSlaveDataLength(uint8_t num, uint8_t length);

// I2C_SLV* registers (Slave 4)
uint8_t getSlave4Address();
        void setSlave4Address(uint8_t address);
        uint8_t getSlave4Register();
        void setSlave4Register(uint8_t reg);
        void setSlave4OutputByte(uint8_t data);
        bool getSlave4Enabled();
        void setSlave4Enabled(bool enabled);
        bool getSlave4InterruptEnabled();
        void setSlave4InterruptEnabled(bool enabled);
        bool getSlave4WriteMode();
        void setSlave4WriteMode(bool mode);
        uint8_t getSlave4MasterDelay();
        void setSlave4MasterDelay(uint8_t delay);
        uint8_t getSlate4InputByte();
```

```
        // I2C_MST_STATUS register
        bool getPassthroughStatus();
        bool getSlave4IsDone();
        bool getLostArbitration();
        bool getSlave4Nack();
        bool getSlave3Nack();
        bool getSlave2Nack();
        bool getSlave1Nack();
        bool getSlave0Nack();

        // INT_PIN_CFG register
        bool getInterruptMode();
        void setInterruptMode(bool mode);
        bool getInterruptDrive();
        void setInterruptDrive(bool drive);
        bool getInterruptLatch();
        void setInterruptLatch(bool latch);
        bool getInterruptLatchClear();
        void setInterruptLatchClear(bool clear);
        bool getFSyncInterruptLevel();
        void setFSyncInterruptLevel(bool level);
        bool getFSyncInterruptEnabled();
        void setFSyncInterruptEnabled(bool enabled);
        bool getI2CBypassEnabled();
        void setI2CBypassEnabled(bool enabled);
        bool getClockOutputEnabled();
        void setClockOutputEnabled(bool enabled);

// INT_ENABLE register
uint8_t getIntEnabled();
        void setIntEnabled(uint8_t enabled);
        bool getIntFreefallEnabled();
        void setIntFreefallEnabled(bool enabled);
        bool getIntMotionEnabled();
        void setIntMotionEnabled(bool enabled);
        bool getIntZeroMotionEnabled();
        void setIntZeroMotionEnabled(bool enabled);
        bool getIntFIFOBufferOverflowEnabled();
        void setIntFIFOBufferOverflowEnabled(bool enabled);
        bool getIntI2CMasterEnabled();
        void setIntI2CMasterEnabled(bool enabled);
        bool getIntDataReadyEnabled();
        void setIntDataReadyEnabled(bool enabled);

// INT_STATUS register
uint8_t getIntStatus();
        bool getIntFreefallStatus();
        bool getIntMotionStatus();
        bool getIntZeroMotionStatus();
        bool getIntFIFOBufferOverflowStatus();
```

```
        bool getIntI2CMasterStatus();
        bool getIntDataReadyStatus();

        // ACCEL_*OUT_* registers
        void getMotion9(int16_t* ax, int16_t* ay, int16_t* az,
int16_t* gx, int16_t* gy, int16_t* gz, int16_t* mx, int16_t* my,
int16_t* mz);
        void getMotion6(int16_t* ax, int16_t* ay, int16_t* az,
int16_t* gx, int16_t* gy, int16_t* gz);
        void getAcceleration(int16_t* x, int16_t* y, int16_t* z);
        int16_t getAccelerationX();
        int16_t getAccelerationY();
        int16_t getAccelerationZ();

        // TEMP_OUT_* registers
        int16_t getTemperature();

        // GYRO_*OUT_* registers
        void getRotation(int16_t* x, int16_t* y, int16_t* z);
        int16_t getRotationX();
        int16_t getRotationY();
        int16_t getRotationZ();

        // EXT_SENS_DATA_* registers
        uint8_t getExternalSensorByte(int position);
        uint16_t getExternalSensorWord(int position);
        uint32_t getExternalSensorDWord(int position);

// MOT_DETECT_STATUS register
bool getXNegMotionDetected();
        bool getXPosMotionDetected();
        bool getYNegMotionDetected();
        bool getYPosMotionDetected();
        bool getZNegMotionDetected();
        bool getZPosMotionDetected();
        bool getZeroMotionDetected();

        // I2C_SLV*_DO register
        void setSlaveOutputByte(uint8_t num, uint8_t data);

// I2C_MST_DELAY_CTRL register
bool getExternalShadowDelayEnabled();
        void setExternalShadowDelayEnabled(bool enabled);
        bool getSlaveDelayEnabled(uint8_t num);
        void setSlaveDelayEnabled(uint8_t num, bool enabled);

// SIGNAL_PATH_RESET register
void resetGyroscopePath();
        void resetAccelerometerPath();
        void resetTemperaturePath();
```

```
// MOT_DETECT_CTRL register
uint8_t getAccelerometerPowerOnDelay();
        void setAccelerometerPowerOnDelay(uint8_t delay);
        uint8_t getFreefallDetectionCounterDecrement();
        void setFreefallDetectionCounterDecrement(uint8_t
decrement);
        uint8_t getMotionDetectionCounterDecrement();
        void setMotionDetectionCounterDecrement(uint8_t
decrement);

// USER_CTRL register
bool getFIFOEnabled();
        void setFIFOEnabled(bool enabled);
        bool getI2CMasterModeEnabled();
        void setI2CMasterModeEnabled(bool enabled);
        void switchSPIEnabled(bool enabled);
        void resetFIFO();
        void resetI2CMaster();
        void resetSensors();

        // PWR_MGMT_1 register
        void reset();
        bool getSleepEnabled();
        void setSleepEnabled(bool enabled);
        bool getWakeCycleEnabled();
        void setWakeCycleEnabled(bool enabled);
        bool getTempSensorEnabled();
        void setTempSensorEnabled(bool enabled);
        uint8_t getClockSource();
        void setClockSource(uint8_t source);

// PWR_MGMT_2 register
uint8_t getWakeFrequency();
        void setWakeFrequency(uint8_t frequency);
        bool getStandbyXAccelEnabled();
        void setStandbyXAccelEnabled(bool enabled);
        bool getStandbyYAccelEnabled();
        void setStandbyYAccelEnabled(bool enabled);
        bool getStandbyZAccelEnabled();
        void setStandbyZAccelEnabled(bool enabled);
        bool getStandbyXGyroEnabled();
        void setStandbyXGyroEnabled(bool enabled);
        bool getStandbyYGyroEnabled();
        void setStandbyYGyroEnabled(bool enabled);
        bool getStandbyZGyroEnabled();
        void setStandbyZGyroEnabled(bool enabled);

        // FIFO_COUNT_* registers
        uint16_t getFIFOCount();

        // FIFO_R_W register
```

```
        uint8_t getFIFOByte();
        void setFIFOByte(uint8_t data);
        void getFIFOBytes(uint8_t *data, uint8_t length);

// WHO_AM_I register
uint8_t getDeviceID();
        void setDeviceID(uint8_t id);

        // ======== UNDOCUMENTED/DMP REGISTERS/METHODS ========

        // XG_OFFS_TC register
        uint8_t getOTPBankValid();
        void setOTPBankValid(bool enabled);
        int8_t getXGyroOffsetTC();
        void setXGyroOffsetTC(int8_t offset);

// YG_OFFS_TC register
int8_t getYGyroOffsetTC();
        void setYGyroOffsetTC(int8_t offset);

// ZG_OFFS_TC register
int8_t getZGyroOffsetTC();
        void setZGyroOffsetTC(int8_t offset);

// X_FINE_GAIN register
int8_t getXFineGain();
        void setXFineGain(int8_t gain);

// Y_FINE_GAIN register
int8_t getYFineGain();
        void setYFineGain(int8_t gain);

// Z_FINE_GAIN register
int8_t getZFineGain();
        void setZFineGain(int8_t gain);

// XA_OFFS_* registers
int16_t getXAccelOffset();
        void setXAccelOffset(int16_t offset);

// YA_OFFS_* register
int16_t getYAccelOffset();
        void setYAccelOffset(int16_t offset);

// ZA_OFFS_* register
int16_t getZAccelOffset();
        void setZAccelOffset(int16_t offset);

// XG_OFFS_USR* registers
int16_t getXGyroOffset();
        void setXGyroOffset(int16_t offset);
```

```
// YG_OFFS_USR* register
int16_t getYGyroOffset();
        void setYGyroOffset(int16_t offset);

// ZG_OFFS_USR* register
int16_t getZGyroOffset();
        void setZGyroOffset(int16_t offset);

        // INT_ENABLE register (DMP functions)
        bool getIntPLLReadyEnabled();
        void setIntPLLReadyEnabled(bool enabled);
        bool getIntDMPEnabled();
        void setIntDMPEnabled(bool enabled);

// DMP_INT_STATUS
bool getDMPInt5Status();
        bool getDMPInt4Status();
        bool getDMPInt3Status();
        bool getDMPInt2Status();
        bool getDMPInt1Status();
        bool getDMPInt0Status();

        // INT_STATUS register (DMP functions)
        bool getIntPLLReadyStatus();
        bool getIntDMPStatus();

        // USER_CTRL register (DMP functions)
        bool getDMPEnabled();
        void setDMPEnabled(bool enabled);
void resetDMP();

        // BANK_SEL register
void setMemoryBank(uint8_t bank, bool prefetchEnabled=false, bool
userBank=false);

// MEM_START_ADDR register
void setMemoryStartAddress(uint8_t address);

// MEM_R_W register
uint8_t readMemoryByte();
        void writeMemoryByte(uint8_t data);
        void readMemoryBlock(uint8_t *data, uint16_t dataSize,
uint8_t bank=0, uint8_t address=0);
        bool writeMemoryBlock(const uint8_t *data, uint16_t
dataSize, uint8_t bank=0, uint8_t address=0, bool verify=true,
bool useProgMem=false);
        bool writeProgMemoryBlock(const uint8_t *data, uint16_t
dataSize, uint8_t bank=0, uint8_t address=0, bool verify=true);
```

```
        bool writeDMPConfigurationSet(const uint8_t *data,
uint16_t dataSize, bool useProgMem=false);
        bool writeProgDMPConfigurationSet(const uint8_t *data,
uint16_t dataSize);

// DMP_CFG_1 register
uint8_t getDMPConfig1();
        void setDMPConfig1(uint8_t config);

// DMP_CFG_2 register
uint8_t getDMPConfig2();
        void setDMPConfig2(uint8_t config);

        // special methods for MotionApps 2.0 implementation
        #ifdef MPU6050_INCLUDE_DMP_MOTIONAPPS20
            uint8_t *dmpPacketBuffer;
            uint16_t dmpPacketSize;

            uint8_t dmpInitialize();
            bool dmpPacketAvailable();

            uint8_t dmpSetFIFORate(uint8_t fifoRate);
            uint8_t dmpGetFIFORate();
            uint8_t dmpGetSampleStepSizeMS();
            uint8_t dmpGetSampleFrequency();
            int32_t dmpDecodeTemperature(int8_t tempReg);

            // Register callbacks after a packet of FIFO data is
processed
            //uint8_t dmpRegisterFIFORateProcess(inv_obj_func
func, int16_t priority);
            //uint8_t dmpUnregisterFIFORateProcess(inv_obj_func
func);
            uint8_t dmpRunFIFORateProcesses();

            // Setup FIFO for various output
            uint8_t dmpSendQuaternion(uint_fast16_t accuracy);
            uint8_t dmpSendGyro(uint_fast16_t elements,
uint_fast16_t accuracy);
            uint8_t dmpSendAccel(uint_fast16_t elements,
uint_fast16_t accuracy);
            uint8_t dmpSendLinearAccel(uint_fast16_t elements,
uint_fast16_t accuracy);
            uint8_t dmpSendLinearAccelInWorld(uint_fast16_t
elements, uint_fast16_t accuracy);
            uint8_t dmpSendControlData(uint_fast16_t elements,
uint_fast16_t accuracy);
            uint8_t dmpSendSensorData(uint_fast16_t elements,
uint_fast16_t accuracy);
            uint8_t dmpSendExternalSensorData(uint_fast16_t
elements, uint_fast16_t accuracy);
```

```
            uint8_t dmpSendGravity(uint_fast16_t elements,
uint_fast16_t accuracy);
            uint8_t dmpSendPacketNumber(uint_fast16_t accuracy);
            uint8_t dmpSendQuantizedAccel(uint_fast16_t elements,
uint_fast16_t accuracy);
            uint8_t dmpSendEIS(uint_fast16_t elements,
uint_fast16_t accuracy);

            // Get Fixed Point data from FIFO
            uint8_t dmpGetAccel(int32_t *data, const uint8_t*
packet=0);
            uint8_t dmpGetAccel(int16_t *data, const uint8_t*
packet=0);
            uint8_t dmpGetAccel(VectorInt16 *v, const uint8_t*
packet=0);
            uint8_t dmpGetQuaternion(int32_t *data, const uint8_t*
packet=0);
            uint8_t dmpGetQuaternion(int16_t *data, const uint8_t*
packet=0);
            uint8_t dmpGetQuaternion(Quaternion *q, const uint8_t*
packet=0);
            uint8_t dmpGet6AxisQuaternion(int32_t *data, const
uint8_t* packet=0);
            uint8_t dmpGet6AxisQuaternion(int16_t *data, const
uint8_t* packet=0);
            uint8_t dmpGet6AxisQuaternion(Quaternion *q, const
uint8_t* packet=0);
            uint8_t dmpGetRelativeQuaternion(int32_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetRelativeQuaternion(int16_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetRelativeQuaternion(Quaternion *data,
const uint8_t* packet=0);
            uint8_t dmpGetGyro(int32_t *data, const uint8_t*
packet=0);
            uint8_t dmpGetGyro(int16_t *data, const uint8_t*
packet=0);
            uint8_t dmpGetGyro(VectorInt16 *v, const uint8_t*
packet=0);
            uint8_t dmpSetLinearAccelFilterCoefficient(float
coef);
            uint8_t dmpGetLinearAccel(int32_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetLinearAccel(int16_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetLinearAccel(VectorInt16 *v, const
uint8_t* packet=0);
            uint8_t dmpGetLinearAccel(VectorInt16 *v, VectorInt16
*vRaw, VectorFloat *gravity);
            uint8_t dmpGetLinearAccelInWorld(int32_t *data, const
uint8_t* packet=0);
```

```
            uint8_t dmpGetLinearAccelInWorld(int16_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetLinearAccelInWorld(VectorInt16 *v, const
uint8_t* packet=0);
            uint8_t dmpGetLinearAccelInWorld(VectorInt16 *v,
VectorInt16 *vReal, Quaternion *q);
            uint8_t dmpGetGyroAndAccelSensor(int32_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetGyroAndAccelSensor(int16_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetGyroAndAccelSensor(VectorInt16 *g,
VectorInt16 *a, const uint8_t* packet=0);
            uint8_t dmpGetGyroSensor(int32_t *data, const uint8_t*
packet=0);
            uint8_t dmpGetGyroSensor(int16_t *data, const uint8_t*
packet=0);
            uint8_t dmpGetGyroSensor(VectorInt16 *v, const
uint8_t* packet=0);
            uint8_t dmpGetControlData(int32_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetTemperature(int32_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetGravity(int32_t *data, const uint8_t*
packet=0);
            uint8_t dmpGetGravity(int16_t *data, const uint8_t*
packet=0);
            uint8_t dmpGetGravity(VectorInt16 *v, const uint8_t*
packet=0);
            uint8_t dmpGetGravity(VectorFloat *v, Quaternion *q);
            uint8_t dmpGetUnquantizedAccel(int32_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetUnquantizedAccel(int16_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetUnquantizedAccel(VectorInt16 *v, const
uint8_t* packet=0);
            uint8_t dmpGetQuantizedAccel(int32_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetQuantizedAccel(int16_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetQuantizedAccel(VectorInt16 *v, const
uint8_t* packet=0);
            uint8_t dmpGetExternalSensorData(int32_t *data,
uint16_t size, const uint8_t* packet=0);
            uint8_t dmpGetEIS(int32_t *data, const uint8_t*
packet=0);

            uint8_t dmpGetEuler(float *data, Quaternion *q);
            uint8_t dmpGetYawPitchRoll(float *data, Quaternion *q,
VectorFloat *gravity);

            // Get Floating Point data from FIFO
```

```
            uint8_t dmpGetAccelFloat(float *data, const uint8_t*
packet=0);
            uint8_t dmpGetQuaternionFloat(float *data, const
uint8_t* packet=0);

            uint8_t dmpProcessFIFOPacket(const unsigned char
*dmpData);
            uint8_t dmpReadAndProcessFIFOPacket(uint8_t
numPackets, uint8_t *processed=NULL);

            uint8_t dmpSetFIFOProcessedCallback(void (*func)
(void));

            uint8_t dmpInitFIFOParam();
            uint8_t dmpCloseFIFO();
            uint8_t dmpSetGyroDataSource(uint8_t source);
            uint8_t dmpDecodeQuantizedAccel();
            uint32_t dmpGetGyroSumOfSquare();
            uint32_t dmpGetAccelSumOfSquare();
            void dmpOverrideQuaternion(long *q);
            uint16_t dmpGetFIFOPacketSize();
        #endif

        // special methods for MotionApps 4.1 implementation
        #ifdef MPU6050_INCLUDE_DMP_MOTIONAPPS41
            uint8_t *dmpPacketBuffer;
            uint16_t dmpPacketSize;

            uint8_t dmpInitialize();
            bool dmpPacketAvailable();

            uint8_t dmpSetFIFORate(uint8_t fifoRate);
            uint8_t dmpGetFIFORate();
            uint8_t dmpGetSampleStepSizeMS();
            uint8_t dmpGetSampleFrequency();
            int32_t dmpDecodeTemperature(int8_t tempReg);

            // Register callbacks after a packet of FIFO data is
processed
            //uint8_t dmpRegisterFIFORateProcess(inv_obj_func
func, int16_t priority);
            //uint8_t dmpUnregisterFIFORateProcess(inv_obj_func
func);
            uint8_t dmpRunFIFORateProcesses();

            // Setup FIFO for various output
            uint8_t dmpSendQuaternion(uint_fast16_t accuracy);
            uint8_t dmpSendGyro(uint_fast16_t elements,
uint_fast16_t accuracy);
            uint8_t dmpSendAccel(uint_fast16_t elements,
uint_fast16_t accuracy);
```

```
            uint8_t dmpSendLinearAccel(uint_fast16_t elements,
uint_fast16_t accuracy);
            uint8_t dmpSendLinearAccelInWorld(uint_fast16_t
elements, uint_fast16_t accuracy);
            uint8_t dmpSendControlData(uint_fast16_t elements,
uint_fast16_t accuracy);
            uint8_t dmpSendSensorData(uint_fast16_t elements,
uint_fast16_t accuracy);
            uint8_t dmpSendExternalSensorData(uint_fast16_t
elements, uint_fast16_t accuracy);
            uint8_t dmpSendGravity(uint_fast16_t elements,
uint_fast16_t accuracy);
            uint8_t dmpSendPacketNumber(uint_fast16_t accuracy);
            uint8_t dmpSendQuantizedAccel(uint_fast16_t elements,
uint_fast16_t accuracy);
            uint8_t dmpSendEIS(uint_fast16_t elements,
uint_fast16_t accuracy);

            // Get Fixed Point data from FIFO
            uint8_t dmpGetAccel(int32_t *data, const uint8_t*
packet=0);
            uint8_t dmpGetAccel(int16_t *data, const uint8_t*
packet=0);
            uint8_t dmpGetAccel(VectorInt16 *v, const uint8_t*
packet=0);
            uint8_t dmpGetQuaternion(int32_t *data, const uint8_t*
packet=0);
            uint8_t dmpGetQuaternion(int16_t *data, const uint8_t*
packet=0);
            uint8_t dmpGetQuaternion(Quaternion *q, const uint8_t*
packet=0);
            uint8_t dmpGet6AxisQuaternion(int32_t *data, const
uint8_t* packet=0);
            uint8_t dmpGet6AxisQuaternion(int16_t *data, const
uint8_t* packet=0);
            uint8_t dmpGet6AxisQuaternion(Quaternion *q, const
uint8_t* packet=0);
            uint8_t dmpGetRelativeQuaternion(int32_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetRelativeQuaternion(int16_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetRelativeQuaternion(Quaternion *data,
const uint8_t* packet=0);
            uint8_t dmpGetGyro(int32_t *data, const uint8_t*
packet=0);
            uint8_t dmpGetGyro(int16_t *data, const uint8_t*
packet=0);
            uint8_t dmpGetGyro(VectorInt16 *v, const uint8_t*
packet=0);
            uint8_t dmpGetMag(int16_t *data, const uint8_t*
packet=0);
```

```
            uint8_t dmpSetLinearAccelFilterCoefficient(float
coef);
            uint8_t dmpGetLinearAccel(int32_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetLinearAccel(int16_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetLinearAccel(VectorInt16 *v, const
uint8_t* packet=0);
            uint8_t dmpGetLinearAccel(VectorInt16 *v, VectorInt16
*vRaw, VectorFloat *gravity);
            uint8_t dmpGetLinearAccelInWorld(int32_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetLinearAccelInWorld(int16_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetLinearAccelInWorld(VectorInt16 *v, const
uint8_t* packet=0);
            uint8_t dmpGetLinearAccelInWorld(VectorInt16 *v,
VectorInt16 *vReal, Quaternion *q);
            uint8_t dmpGetGyroAndAccelSensor(int32_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetGyroAndAccelSensor(int16_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetGyroAndAccelSensor(VectorInt16 *g,
VectorInt16 *a, const uint8_t* packet=0);
            uint8_t dmpGetGyroSensor(int32_t *data, const uint8_t*
packet=0);
            uint8_t dmpGetGyroSensor(int16_t *data, const uint8_t*
packet=0);
            uint8_t dmpGetGyroSensor(VectorInt16 *v, const
uint8_t* packet=0);
            uint8_t dmpGetControlData(int32_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetTemperature(int32_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetGravity(int32_t *data, const uint8_t*
packet=0);
            uint8_t dmpGetGravity(int16_t *data, const uint8_t*
packet=0);
            uint8_t dmpGetGravity(VectorInt16 *v, const uint8_t*
packet=0);
            uint8_t dmpGetGravity(VectorFloat *v, Quaternion *q);
            uint8_t dmpGetUnquantizedAccel(int32_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetUnquantizedAccel(int16_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetUnquantizedAccel(VectorInt16 *v, const
uint8_t* packet=0);
            uint8_t dmpGetQuantizedAccel(int32_t *data, const
uint8_t* packet=0);
            uint8_t dmpGetQuantizedAccel(int16_t *data, const
uint8_t* packet=0);
```

```
            uint8_t dmpGetQuantizedAccel(VectorInt16 *v, const
uint8_t* packet=0);
            uint8_t dmpGetExternalSensorData(int32_t *data,
uint16_t size, const uint8_t* packet=0);
            uint8_t dmpGetEIS(int32_t *data, const uint8_t*
packet=0);

            uint8_t dmpGetEuler(float *data, Quaternion *q);
            uint8_t dmpGetYawPitchRoll(float *data, Quaternion *q,
VectorFloat *gravity);

            // Get Floating Point data from FIFO
            uint8_t dmpGetAccelFloat(float *data, const uint8_t*
packet=0);
            uint8_t dmpGetQuaternionFloat(float *data, const
uint8_t* packet=0);

            uint8_t dmpProcessFIFOPacket(const unsigned char
*dmpData);
            uint8_t dmpReadAndProcessFIFOPacket(uint8_t
numPackets, uint8_t *processed=NULL);

            uint8_t dmpSetFIFOProcessedCallback(void (*func)
(void));

            uint8_t dmpInitFIFOParam();
            uint8_t dmpCloseFIFO();
            uint8_t dmpSetGyroDataSource(uint8_t source);
            uint8_t dmpDecodeQuantizedAccel();
            uint32_t dmpGetGyroSumOfSquare();
            uint32_t dmpGetAccelSumOfSquare();
            void dmpOverrideQuaternion(long *q);
            uint16_t dmpGetFIFOPacketSize();
        #endif

    private:
        uint8_t devAddr;
        uint8_t buffer[14];
};

#endif /* _MPU6050_H_ */
```

Disponible en: https://github.com/jrowberg/i2cdevlib/tree/master/Arduino.

ANEXO X. CODIGO CARGA PAGA

```
#include <SFE_BMP180.h>
#include <Wire.h>
#include <SD.h>

File myFile;
SFE_BMP180 bmp180;

double Po;
char status;
double T,P,A,h;
double PresionNivelMar=1013.25;
void setup()
{
Serial.begin(9600);
   Serial.print("Iniciando SD ...");
  if (!SD.begin(4)) {
Serial.println("No se pudo inicializar");
return;
    }
    else
    {
    Serial.println("inicializacion exitosa");
  }

if (bmp180.begin())  {
Serial.println("BMP180 iniciado correctamentenTomando medidadas del punto de
referncia...n");
status = bmp180.startTemperature();
    if (status != 0)
    {
      delay(status);
status = bmp180.getTemperature(T);
      if (status != 0)
      {
status = bmp180.startPressure(3);
if (status != 0)
        {
          delay(status);//Pausa para que finalice la lectura
status = bmp180.getPressure(P,T);
```

```
if (status != 0)
        {
          Po=P;
          Serial.println("Punto de referncia establecido: h=0");
        }
       }
     }
   }

  }
  else
  {
    Serial.println("Error al iniciar el BMP180");
while(1);
  }
}

void loop()
{


  status = bmp180.startTemperature();
if (status != 0)
  {
    delay(status); //Pausa para que finalice la lectura
status = bmp180.getTemperature(T);
    if (status != 0)
    {
      status = bmp180.startPressure(3);
      if (status != 0)
      {
        delay(status);//Pausa para que finalice la lectura
status = bmp180.getPressure(P,T);
if (status != 0)
       {
Serial.print("Temperatura: ");
          Serial.print(T);
          Serial.print(" *C , ");
          Serial.print("Presion: ");
          Serial.print(P);
Serial.print(" mb , ");
A= bmp180.altitude(P,PresionNivelMar);
```

```
        Serial.print("Altitud: ");
        Serial.print(A);
Serial.print(" m s.n.m. ");
h= bmp180.altitude(P,Po);
Serial.print("Altitud desde Bogota ");
Serial.print(h);
        Serial.println(" metros");
}
     }
   }
  }
myFile = SD.open("datalog.txt", FILE_WRITE);

  if (myFile) {
Serial.print("Escribiendo SD: ");
        myFile.print("Temperatura: ");
myFile.print(T);
        myFile.print(" *C , ");
        myFile.print("Presion: ");
        myFile.print(P);
        myFile.print(" mb , ");
        myFile.print("Altitud: ");
myFile.print(A);
myFile.print(" m s.n.m. ");
myFile.print("Altitud sobre Referencia");
        myFile.print(h);
        myFile.println(" metros");

       myFile.close();


} else {
    Serial.println("Error al abrir el archivo");
  }
  delay(1000);
}
```

ANEXO Y. CODIGO DE SISTEMA DE RECUPERACIÓN

```
#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"
#include <SD.h>
File myFile;
MPU6050 sensor;
int ax, ay, az;
int gx, gy, gz;
uint16_t i=0;
int lee=7;
int fijo=9;
int motor_arriba=5;
int motor_abajo=8;
void setup() {
 Serial.begin(9600);
 Wire.begin();
 sensor.initialize();
 if (sensor.testConnection()) Serial.println("Sensor iniciado correctamente");
 else Serial.println("Error al iniciar el sensor");
 Serial.print("Iniciando SD ...");
 pinMode(motor_arriba,OUTPUT);
 pinMode(motor_abajo,OUTPUT);

}
void loop() {
 sensor.getAcceleration(&ax, &ay, &az);
 sensor.getRotation(&gx, &gy, &gz);
 float ax_m_s2 = ax * (9.81/16384.0);
 float ay_m_s2 = ay * (9.81/16384.0);
 float az_m_s2 = az * (9.81/16384.0);
 float gx_deg_s = gx * (250.0/32768.0);
 float gy_deg_s = gy * (250.0/32768.0);
 float gz_deg_s = gz * (250.0/32768.0);
191
 float accel_ang_x=atan(ax/sqrt(pow(ay,2) + pow(az,2)))*(180.0/3.14);
 float accel_ang_y=atan(ay/sqrt(pow(ax,2) + pow(az,2)))*(180.0/3.14);
 Serial.print("t ");
 Serial.print(i++);
 Serial.print(" ");
```

```
 Serial.print(ax_m_s2); Serial.print(" ");
 Serial.print(" ");
 Serial.print(ay_m_s2); Serial.print(" ");
 Serial.print(" ");
 Serial.print(az_m_s2); Serial.print(" ");
 Serial.print(gx_deg_s); Serial.print(" ");
 Serial.print(gy_deg_s); Serial.print(" ");
 Serial.println(gz_deg_s);
 Serial.print("Inclinacion en X: ");
 Serial.print(accel_ang_x);
 Serial.print("tInclinacion en Y:");
 Serial.println(accel_ang_y);
 myFile = SD.open("datalog.txt", FILE_WRITE);
 if (myFile) {
 myFile.print("t ");
 myFile.print(i++);
 myFile.print (" ");
 myFile.print (ax_m_s2); Serial.print(" ");
 myFile.print (" ");
 myFile.print (ay_m_s2); Serial.print(" ");
 myFile.print (" ");
 myFile.print (az_m_s2); Serial.print(" ");
 myFile.print (gx_deg_s); Serial.print(" ");
 myFile.print (gy_deg_s); Serial.print(" ");
 myFile.print(gz_deg_s);
 myFile.print("Inclinacion en X: ");
 myFile.print(accel_ang_x);
 myFile.print ("tInclinacion en Y:");
 myFile.println(accel_ang_y);
 myFile.close();
 if ((accel_ang_x > 75) || (accel_ang_x < -75)){
 digitalWrite (motor_arriba, HIGH);
192
 delay (3000);
 digitalWrite (motor_arriba, LOW);
 digitalWrite (motor_abajo, HIGH);
 delay (3000);
 digitalWrite (motor_abajo, LOW);
 for(;;){}
 } else {
 digitalWrite (motor_arriba, LOW);
 digitalWrite (motor_abajo, LOW);
```

```
 }
 if((accel_ang_y > -15) || (i==150)){
 digitalWrite (motor_arriba, HIGH);
 delay (3000);
 digitalWrite (motor_arriba, LOW);
 digitalWrite (motor_abajo, HIGH);
 delay (3000);
 digitalWrite (motor_abajo, LOW);
 for(;;){}

 } else {
 digitalWrite (motor_arriba, LOW);
 digitalWrite (motor_abajo, LOW);
 }
 delay(200);
}
```