UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA AERONÁUTICA Y DEL ESPACIO

GRADO EN INGENIERÍA AEROESPACIAL

TRABAJO FIN DE GRADO

Diseño preliminar de aeronaves no tripuladas no convencionales (RPAS), de tamaño reducido

AUTOR: Enrique González Morales

ESPECIALIDAD: Vehículos Aeroespaciales (VA)

TUTOR DEL TRABAJO: Ángel Antonio Rodríguez Sevillano

Julio de 2018

Contenido

Ilustraciones:

1 Introducción

Los RPAS (Remotely piloted aircraft sustems) se caracterizan por ser aeronaves pilotadas remotamente o de forma autónoma. Los RPAS ya son de uso común y están experimentando un gran desarrollo y aumento de demanda en los últimos años.

Los RPAS atraen por sus bajos costes iniciales, de mantenimiento y de operación, pero ofrecen mucha versatilidad, como soporte a tareas de rescate, vigilancia, filmación, recreativas…

Pueden encontrarse en diferentes tamaños y pesos. Una primera clasificación sería la siguiente, según *Barcala Montejano & Rodríguez Sevillano, 2015*:

Categoría de RPAS	Rango (km)	Altura (m)	Envergadura (h)	Peso (kg)
Estratosféricos	> 2.000	20.000 – 30.000	48	< 3.000
Gran altitud y larga duración (HALE)	> 2.000	20.000	48	15.000
Media altitud y larga duración (MALE)	> 500	14.000	24 – 48	1.500
Baja altitud y larga duración (LALE)	> 500	3.000	Aprox 24	Aprox 30
Baja altitud y amplia penetración (LAPD)	> 250	50 - 9.000	0,25 – 1	350
Medio alcance	70 a 500	8.000	6 – 18	1.250
Corto alcance	10 -70	3.000	3 – 6	200
Mini	< 10	< 300	< 2	< 30
Micro	< 10	< 250	1	< 1

En la naturaleza podemos encontrar un ejemplo muy parecido a lo que buscamos en un RPAS, los pájaros. Distintas aves han adaptado su morfología para obtener las características que queremos en un RPAS como alcance y/o autonomía máxima, vuelo a baja velocidad, peso ligero, cargas de pago altas y mucha maniobrabilidad.

Podemos buscar soluciones de diseño de RPAS en aves que con características similares a las nuestras obtengan los resultados que nosotros queremos.

1.1 Motivación

Muchas zonas extensas se pueden beneficiar de vigilancia y monitorización ambiental permanente. Algunos ejemplos son vigilancia de zonas en conflicto bélico, control de incendios en áreas forestales, seguridad en eventos multitudinarios, control de accesos en fronteras…

Para poder dar una vigilancia permanente es necesario una aeronave que pueda planear durante largos periodos de tiempo usando el motor lo más mínimo posible.

Las aves tienen la capacidad de permanecer largos periodos de tiempo volando sin necesidad de bajar a tierra. Por lo que se buscará una solución biométrica basada en aves para diseñar la aeronave.

Para estudiar la sustentación y resistencia del RPAS se desarrollará un programa que combinará XFOIL y VLM para obtener los coeficientes de sustentación y resistencia.

2 Requerimientos

Para vigilar zonas extensas se presentan dos opciones:

- Una aeronave de vuelo a gran altura, veloz, pesada y con cámaras de muy alta resolución.
- Enjambre de pequeñas aeronaves volando a baja altura, baja velocidad, extremadamente ligeras y baratas.

En cualquiera de los dos casos el requerimiento principal es ser capaz de planear durante largos periodos de tiempo.

Entre las dos opciones elegimos la segunda, un enjambre de pequeños RPAS, por una serie de razones:

- Muchos RPAS en una zona bélica permiten que la pérdida de una aeronave no sea un problema, dado que su carga de trabajo se puede repartir en el resto de aeronaves.
- El coste de pérdida o avería de una aeronave es mínimo y son fácil y rápidamente sustituibles.
- En caso de que la aeronave se caiga, el posible daño personal y medioambiental es mínimo, importante si se quiere volar encima de multitudes.
- Permite adquirir datos ambientales en múltiples puntos en vez de uno solo.
- El coste derivado de manejar muchos RPAS al mismo tiempo es fácilmente evitable con pilotos automáticos y control por GPS.

Se decide optar por un micro RPAS, según la clasificación presentada en la introducción, de un metro de envergadura, 800 g de peso y que sea capaz de planear a nivel del mar durante largos periodos de tiempo a velocidades cercanas a 10 m/s.

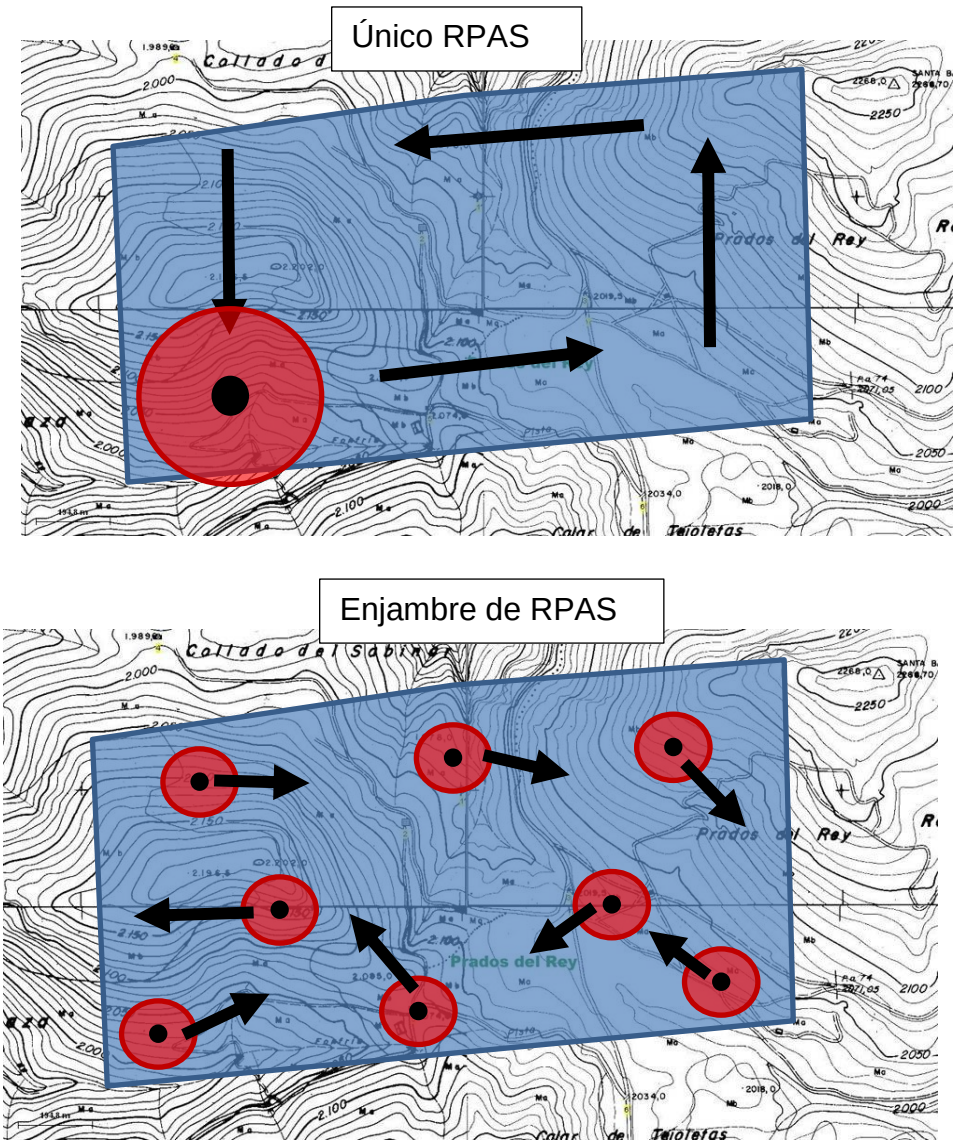


Ilustración 1: Comparación de distintas opciones de vigilancia.

3 Estudio de aeronaves similares.

3.1 Aeronaves similares.

Si buscamos aeronaves con pesos y tamaños similares podemos encontrarlas en aeronaves de uso militar y en aeronaves de uso recreativo. La mayoría de los RPAS para uso militar son de mayor tamaño, pero algunos de los utilizados en misiones de vigilancia tienen tamaños y pesos propios de un micro RPAS. En el uso recreativo se pueden encontrar aeronaves de radiocontrol para aficionados de tamaño y peso similares cuyo objetivo es volar durante largos periodos de tiempo.

3.1.1 AeroVironment Wasp III

Ilustración 2: AeroVironment Wasp III

Tiene una configuración de ala volante y un peso incluso inferior al que queremos conseguir. Se utiliza en fuerzas militares de varios países, entre ellos en el Ejército del Aire español. Puede ser desarmado, llevado en una mochila y utiliza un motor eléctrico y baterías, por lo que es muy versátil. Lleva dos cámaras equipadas para dar soporte de inteligencia y vigilancia. Algunas de sus principales características son:

Envergadura	Peso	Velocidad	Alcance	Autonomía
723 mm	430 g	65 km/h	5 km	45 min

3.1.2 Parrot disco FPV.

Ilustración 3: Parrot Disco FPV

De uso recreativo para vuelo en primera persona. Hecho de carbono y PPE (Polipropileno Expandido), materiales ligeros y comunes en pequeños RPAS que se pueden tomar como ejemplo.

Envergadura	Peso	Velocidad	Rango	Autonomía
1150 mm	750 g	80 km/h	2 km	45 min

3.1.3 Lehmann Aviation LM450

Ilustración 4: Lehmann Aviation LM450

De uso civil, profesional y recreativo, principalmente para tareas de transmisión en vídeo en tiempo real. De lanzamiento manual, muy similar en tamaño y peso al diseño que queremos conseguir.

Envergadura	Peso	Velocidad	Alcance	Autonomía
920 mm	950 g	25 km/h	5 km	45 min

3.1.4 ZALA 421-08

Ilustración 5: ZALA 421-08

De diseño y fabricación rusa. Principalmente usado para tareas de reconocimiento. De lanzamiento manual, aterrizaje por paracaídas y motor eléctrico.

Envergadura	Peso	Velocidad	Rango	Autonomía
800 mm	1700 g	65 km/h	15 km	90 min

4 Estimación de pesos.

Del estudio de aeronaves similares se deduce que podemos conseguir un peso de 800g para una envergadura de 1 metro. Tomando ese peso como el total se decide estudiar la distribución de pesos utilizando como referencia componentes comunes de aeronaves de radiocontrol.

	Masa (g)
Carga de pago	172
Raspberry Pi 3B+	60
Servo (x2)	22
Receptor	10
ESC	36
Motor	50
Batería	150
Cuerpo (PPE)	300
Total	800

Se incluye una Raspberry Pi como ejemplo de pequeño ordenador que permita incluir piloto automático y capacidad para recolectar e interpretar datos si se requiere. Los 172 g de carga de pago son suficientes para añadir una pequeña batería secundaria y multitud de módulos de Raspberry Pi. Los módulos de Raspberry Pi añaden cámaras de baja resolución, GPS, sensores de presión, humedad, temperatura, radiación, aceleración...

5 Código para el cálculo de la polar.

El método V.L.M (Vortex Lattice Method) es un método numérico de dinámica de fluidos computacional. El V.L.M es una extensión de la teoría del hilo sustentador (LLT) de Prandtl, con la diferencia de que, en vez de tratar con un torbellino por ala, se trabaja con una malla de torbellinos. Es un método de flujo potencial, una simplificación del flujo real que permite dimensionar alas rápidamente. Algunos de los problemas que presenta este método es que no tiene en cuenta ni el espesor ni los efectos viscosos y solo se puede utilizar para pequeños ángulos de ataque.

XFOIL es un programa de los años 80 desarrollado por Mark Drela en el MIT que permite estudiar perfiles aerodinámicos aislados en régimen subsónico. Permite estudiar la polar del perfil teniendo en cuenta los efectos viscosos. Es ampliamente conocido por estudiantes y profesionales del sector.

Usando XFOIL y V.L.M. juntos se quiere suplir las deficiencias de espesor y viscosidad que presenta el VLM con los resultados del XFOIL.

Para ello se utilizará primero XFOIL para estudiar la polar del perfil 2D y el VLM (Vortex Lattice Method) para estudiar el ala.

Python es un lenguaje de programación interpretado, multiplataforma y de código abierto que busca ser lo más legible y accesible posible. En los últimos años se está extendiendo en el mundo profesional aeroespacial y científico, pero sobre todo al educativo. Por todo lo descrito anteriormente se elige Python como lenguaje para escribir este código.

Se utilizará Python 3.6 en la plataforma Jupiter Notebook 5.4.0 que permite trabajar cómoda y rápidamente.

5.1 Descripción del código.

5.1.1 Interacción con XFOIL.

Se quiere hacer todo el trabajo a través de Python sin necesidad de utilizar de forma manual otros programas. Por eso dedicamos la primera parte del código para interactuar con XFOIL a través de Python.

Las entradas del programa son las mismas que las del uso convencional de XFOIL, el perfil NACA que se desea estudiar, el número de Reynolds, el número de Mach y el rango de ángulos de ataque que se quiere estudiar. El resultado es un archivo de .txt fácilmente importable a Python. Esta primera parte del código se puede encontrar en el anexo 1.

La segunda parte del código importa e interpreta los datos del .txt generado por XFOIL. Como resultado obtiene Cl(α), Cd(α), Cd(Cl), α0 y Clα. Esta segunda parte del código se puede encontrar en el anexo 2.

El siguiente diagrama de flujo representa las dos primeras partes del código:

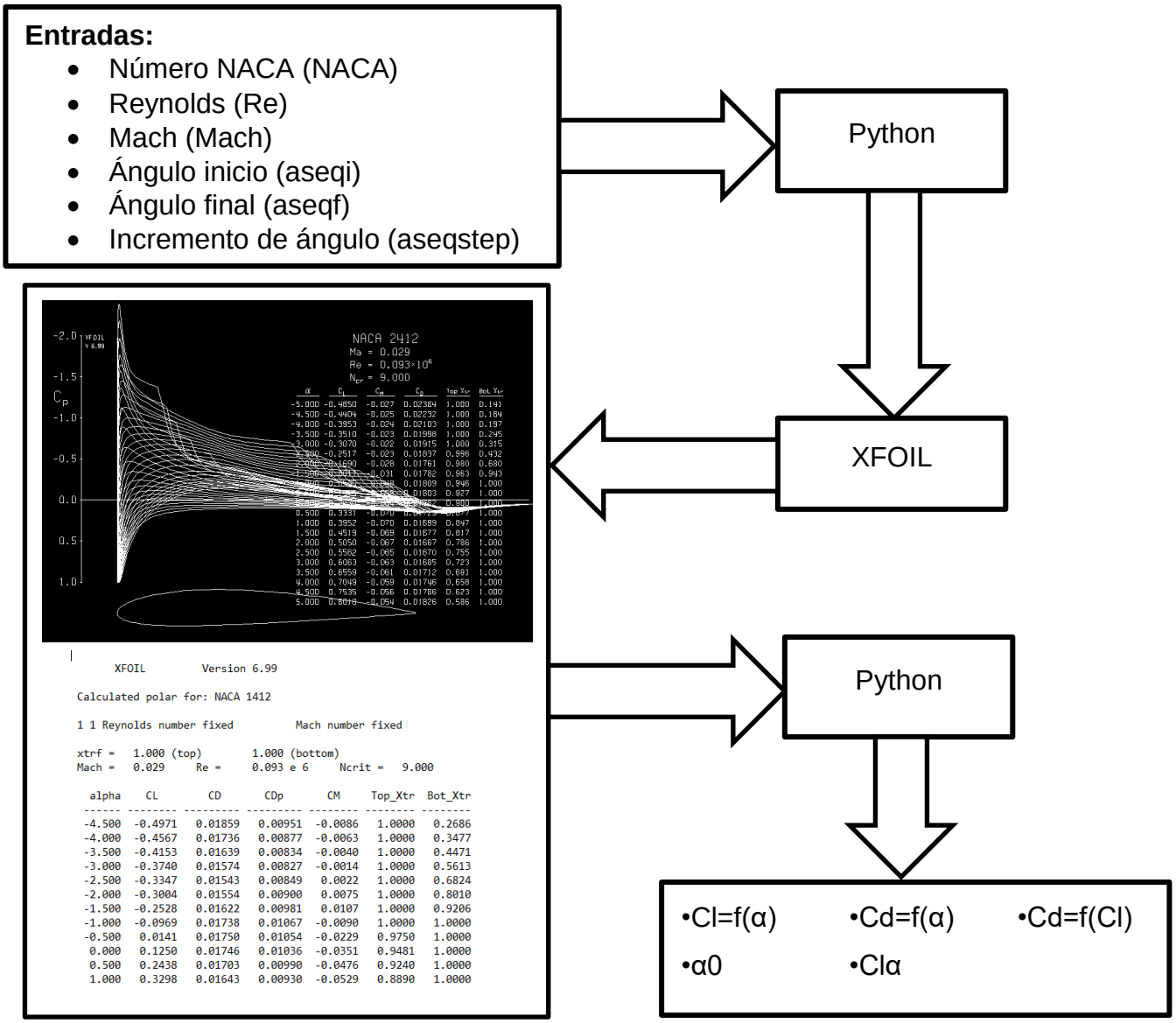


Ilustración 6: Diagrama de flujo de las dos primeras partes del código.

5.1.2 Implementación del VLM.

Para implementar el VLM se utiliza una simplificación para alas planas. Por ser un ala 2D se tomará solo la componente z de la velocidad inducida en cada panel:

$$\overrightarrow{V_{(i,J)}} \approx w_v \vec{k}$$

Se modifica la formulación de la velocidad inducida en el punto de control de cada panel m por el panel n añadiendo la pendiente de la parte linear de la polar del perfil, Cl_α:

$$[w] = [\mathrm{C}] \cdot [\Gamma]$$

$$[C] = C_{m,n}\big|_{derecha} + C_{m,n}\big|_{izquierda}$$

$$C_{m,n} = \frac{1}{2Cl_\alpha}\left[\frac{1}{ad-cb}\left(\frac{ga+hb}{e} - \frac{gc+hd}{f}\right) - \frac{1}{b}\left(1+\frac{a}{e}\right) + \frac{1}{d}\left(1+\frac{c}{f}\right)\right]$$

siendo:

$$a = x_m - x_{1n} \qquad b = y_m - y_{1n}$$

$$e = \sqrt{a^2 + b^2} \qquad f = \sqrt{c^2 + d^2}$$

$$g = x_{2n} - x_{1n} \qquad h = y_{2n} - y_{1n}$$

$$c = x_m - x_{2n} \qquad d = y_m - y_{2n}$$

Los subíndices 1 y 2 se refieren a las esquinas izquierda y derecha del torbellino de herradura en ambas semialas. El subíndice m representa el punto de control del panel m.

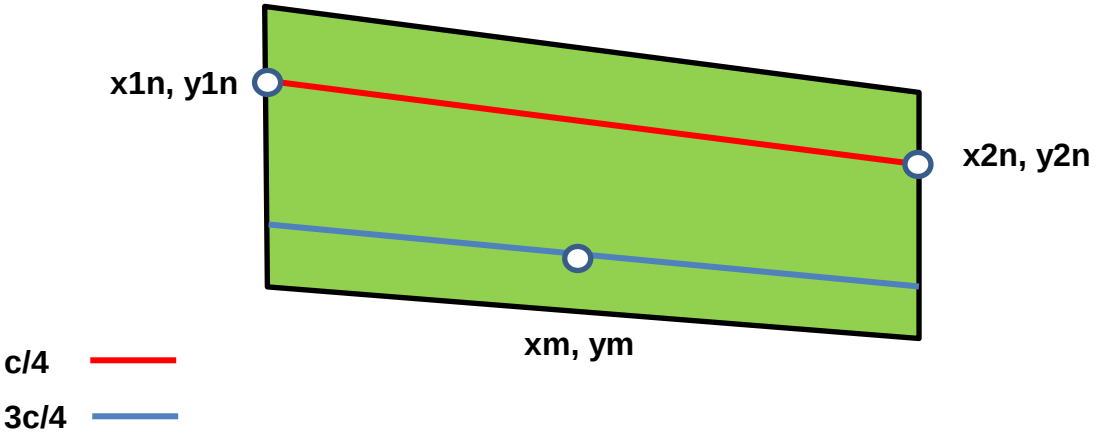


Ilustración 7: Distribución de los puntos en un panel derecho típico.

Siendo la condición de contorno:

$$w_m \approx -V_\infty \cdot \alpha$$

Resolvemos el sistema matricial de forma:

$$[\Gamma] = [C]^{-1} \cdot [w]$$

Por lo que el coeficiente de sustentación del panel:

$$Cl_n = \frac{2 \cdot \Gamma_n}{V_\infty \cdot c_n} - \alpha_0 \cdot Cl_\alpha$$

Donde c_n es la cuerda en el punto de control de cada panel, V_∞ la velocidad de incidencia del aire y α_0 el ángulo de sustentación nula del perfil calculado en XFOIL.

Para el cálculo de Cd_n se utiliza la función $Cd = f(Cl)$ del perfil obtenida mediante XFOIL, introduciendo como Cl el Cl_n de cada panel.

Para el cálculo del coeficiente de momentos es necesario calcular primero la cuerda media de la aeronave:

$$\bar{c} = \frac{1}{S} \cdot \int_{-b/2}^{b/2} c^2(y) \cdot dy$$

El coeficiente de momentos de cada panel se puede calcular:

$$Cm_{OYn} = C_{ln} \cdot \frac{(x_{c/4})_n}{\bar{c}}$$

Siendo $(x_{c/4})_n$ la posición de la línea de un cuarto de cuerda en el punto de control.

Se calculan C_L , C_D y C_{MOY} del ala sumando los efectos de cada panel:

$$C_L = \frac{2}{S} \sum_{n=1}^{n=N} Cl_n \cdot b_n \cdot c_n$$

$$C_D = \frac{2}{S} \sum_{n=1}^{n=N} Cd_n \cdot b_n \cdot c_n$$

$$C_{MOY} = \frac{2}{S} \sum_{n=1}^{n=N} Cm_{OYn} \cdot b_n \cdot c_n$$

Donde b_n es la envergadura de cada panel.

La distribución de los paneles será 1 por distribución de cuerda y 8 por cada semiala, todos con la misma envergadura. Se puede utilizar otras distribuciones en la envergadura, pero es importante utilizar solo un panel por cuerda.

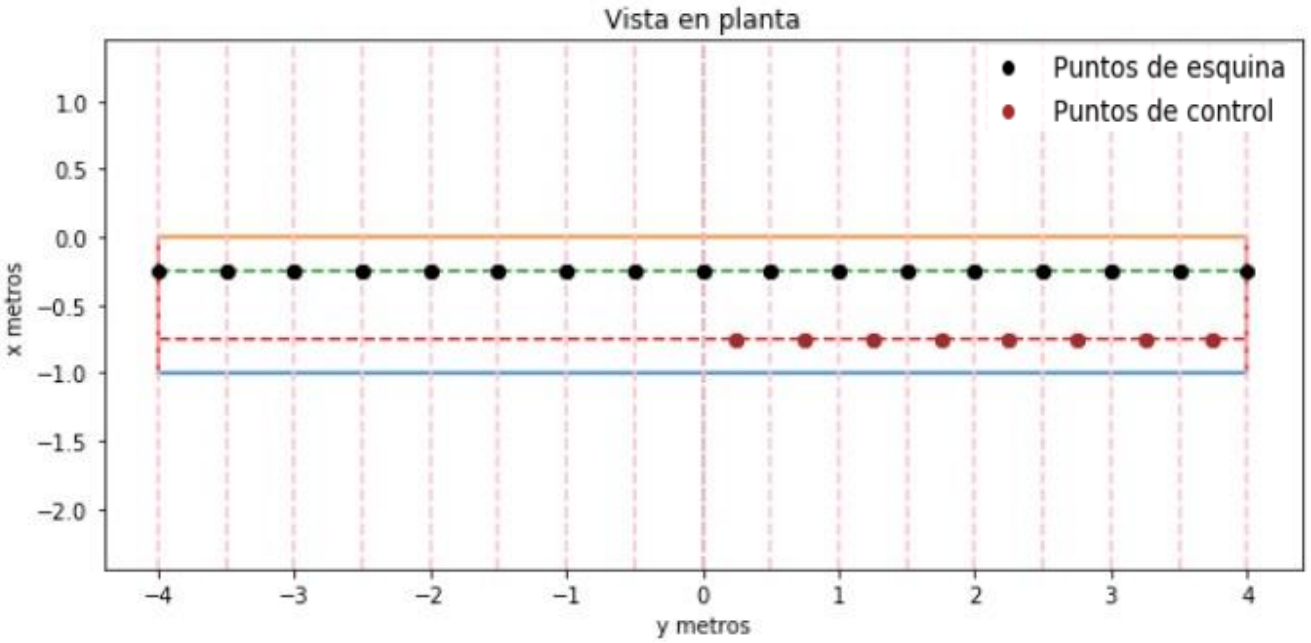


Ilustración 8: Ejemplo de distribución de paneles.

El código completo se puede encontrar en el anexo 3.

5.1.3 Código para la representación 3-D

Un código independiente permite la representación 3-D de la aeronave en Python, utilizando la librería matplotlib sin necesidad de recurrir a otro programa. Se trabaja de forma independiente porque requiere muchos recursos y ralentiza el resto del código. Esta representación es aproximada y solo se utilizará para visualizar el RPAS. La representación 3-D está limitada a perfiles NACA de 4 cifras.

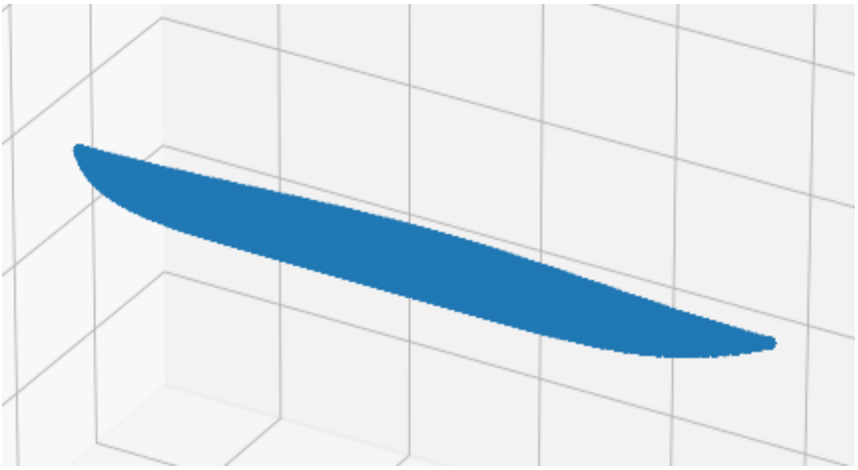


Ilustración 9:Ejemplo del resultado del código para representación 3-D

5.2 Verificación del código.

Para verificar los resultados obtenidos por el código se utilizan una serie de ejemplos con resultados conocidos. El cálculo del resto de coeficientes deriva del cálculo del coeficiente de sustentación por lo que es lo que se usa para verificar el código. En todos los casos el error relativo es menor del 3% por lo que se da el método como válido.

5.2.1 Bertin Smith 2-D Wing

Ala 2-D del libro Aerodynamics for engineers de John J. Bertin y Michael L. Smith capítulo 7, ejemplo del método VLM.

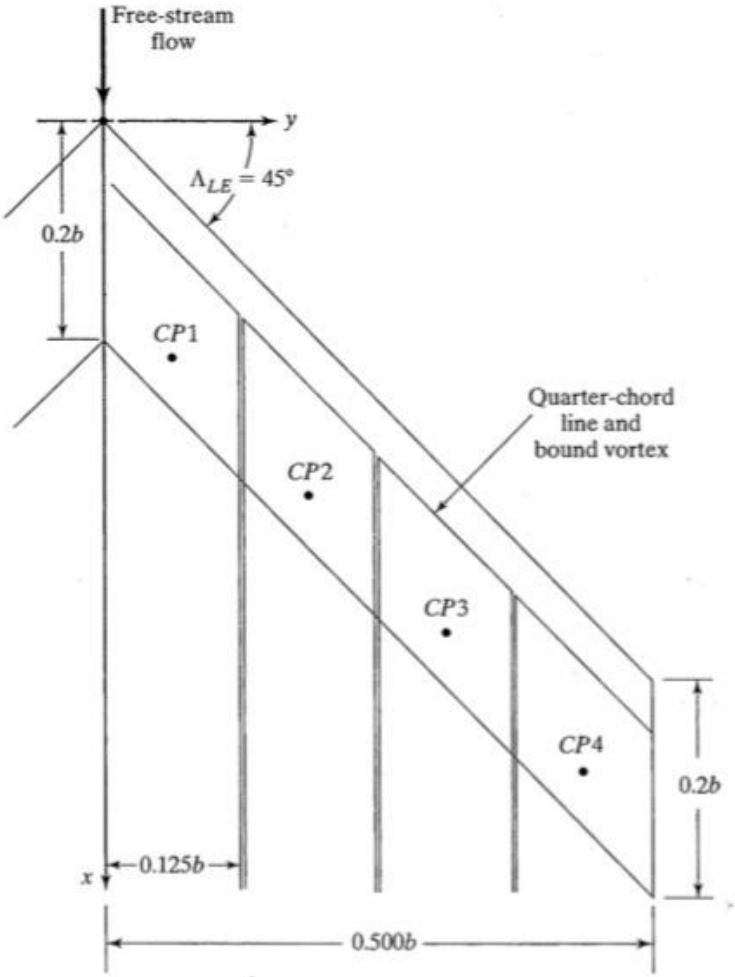


Figure 7.31 Four-panel representation of a swept planar wing, taper ratio of unity, AR = 5, Λ = 45°.

Ilustración 10: Bertin Smith 2-D Wing

	Bertin-Smith	Código TFG
CLα (por radián)	3.433	3.444

5.2.2 Comparación con software Surfaces y Method 1 USAF DATCOM

Comparación obtenida del manual de usuario del software Surfaces de Great OWL Publishing.

A continuación se describe el Method 1 USAF DATCOM para obtener el coeficiente de sustentación:

$$C_{L_\alpha} = \frac{2\pi \cdot AR}{2 + \sqrt{\frac{AR^2\beta^2}{\kappa^2}\left(1 + \frac{\tan^2 \Lambda_{C/2}}{\beta^2}\right) + 4}}$$

$$\beta = \sqrt{1 - M^2} \qquad \kappa = Cl\alpha/2\pi$$

Primera ala:

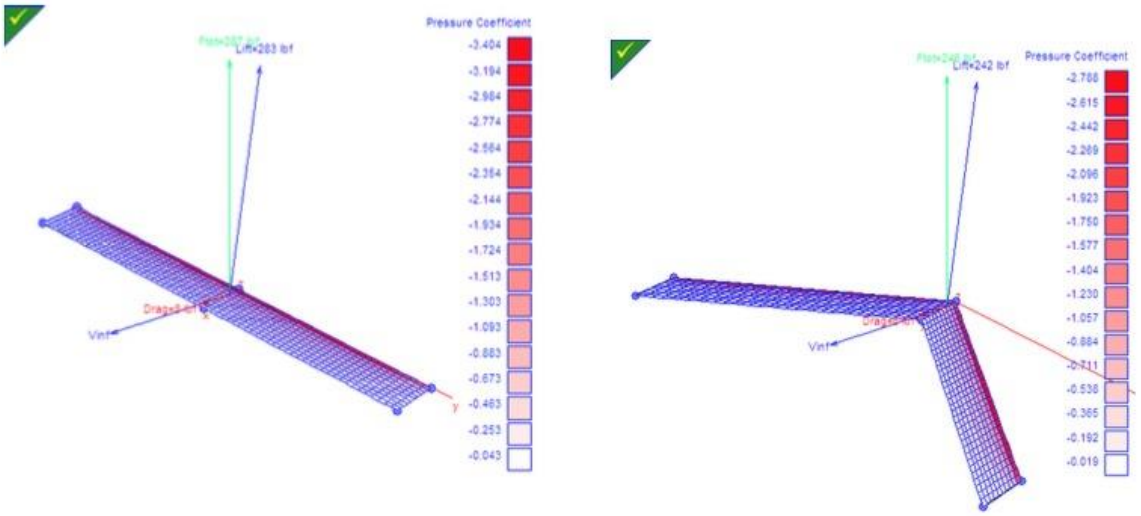


Ilustración 11: Alas con 0 y 35º de flecha.

Características del ala y del flujo:				
b=10ft	c=1ft	V=168,8 ft/s	AR=10	M=0.151
$C_{L\alpha 2D}$=0.1063	S=10ft^2	β=0.989	κ=0.969	$\Lambda_{c/2}$=0º y 35º
ρ=0.002378 slugs/ft^3				

Resultados:			
CLα (por radián)	SURFACES	Method 1	Código TFG
$\Lambda_{c/2}$=0º	4.927	5.086	4.962
$\Lambda_{c/2}$=35º	4.220	4.286	4.277

Segunda ala, rectangular de alargamiento 20:

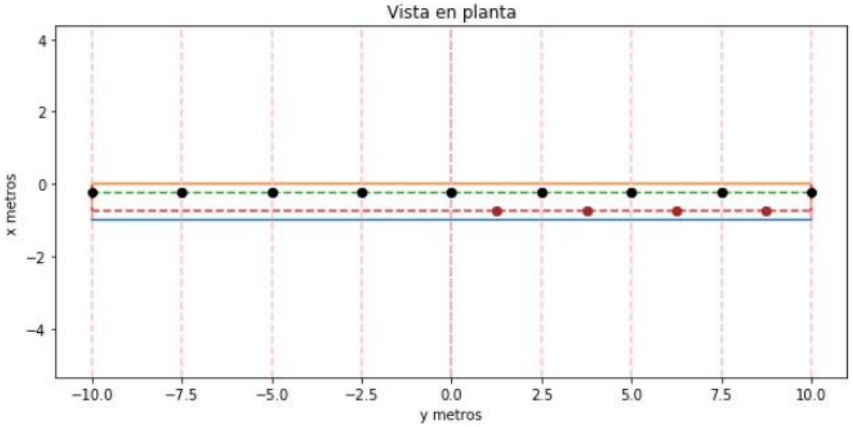


Ilustración 12: Ala rectangular AR=20

Características del ala y del flujo:				
b=20ft	c=1ft	V=100 ft/s	AR=20	M=1
$C_{L\alpha 2D}$=0.1063	S=10ft^2	β=0.989	κ=0.958	$\Lambda_{c/2}$=0º y 35º
ρ=0.002378 slugs/ft^3				

Resultados:			
	SURFACES	Method 1	Código TFG
CLα (por radián)	5.40	5.47	5.43

5.2.3 Warren 12 Wing

Ala típica para verificar los resultados de programas de VLM.

$$AR = 2\sqrt{2}$$
$$\Lambda_{LE} = 53.54°$$
$$S_{wing} = 2\sqrt{2}$$
$$C_{L_\alpha} = 2.743 / rad$$
$$C_{m_\alpha} = -3.10 / rad$$

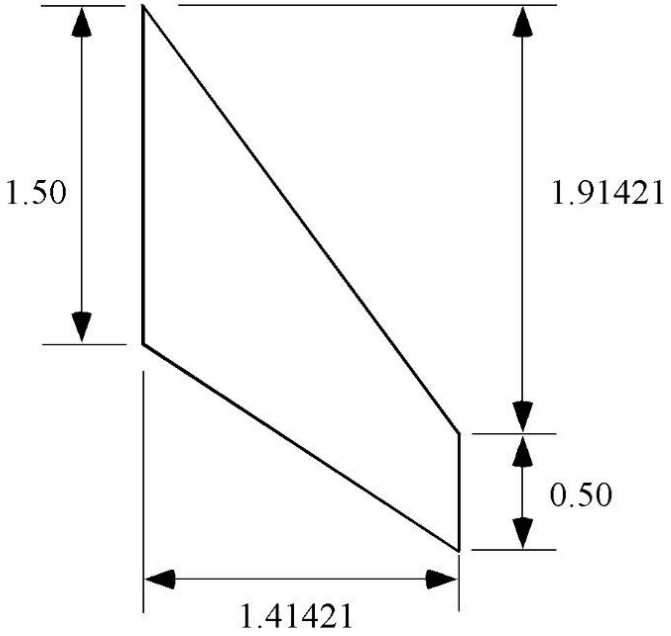


Ilustración 13: Warren 12 wing.

	Warren 12 Wing	Código TFG
CLα (por radián)	2.743	2.75

6 Definición de la geometría alar.

Vamos a basar la geometría alar en la de un ave existente. Las aves tienen la capacidad de permanecer largos periodos de tiempo volando sin necesidad de bajar a tierra. Hay pájaros como el vencejo o el albatros que pueden permanecer meses sin aterrizar. Esto lo consiguen planeando a baja velocidad y aprovechando las corrientes de aire. Por esto se decide buscar como solución a la geometría del ala un diseño biométrico basado en pájaros.

Principalmente hay dos estrategias que han seguido las aves para conseguir planear durante largos periodos de tiempo, alas grandes que conllevan mucho peso o alas pequeñas pero muy ligeras. Para el diseño de micro RPAS reducido nos interesan estos últimos.

6.1 Elección del ave a replicar.

Se va a plantear como objetivo buscar un ave que sea capaz de planear durante largos periodos de tiempo y se va a extrapolar su configuración alar a un RPAS de tamaño reducido.

Los requisitos de la aeronave son 1 metro de envergadura, 800 gramos de peso y capacidad de planear durante largos periodos de tiempo

Para encontrar un ave que sea capaz de planear largas distancias durante largos periodos de tiempo se recurre al listado de aves migratorias planeadoras del estrecho de Gibraltar. Para comparar las distintas aves se comparan envergadura, masa y la proporción entre ambas.

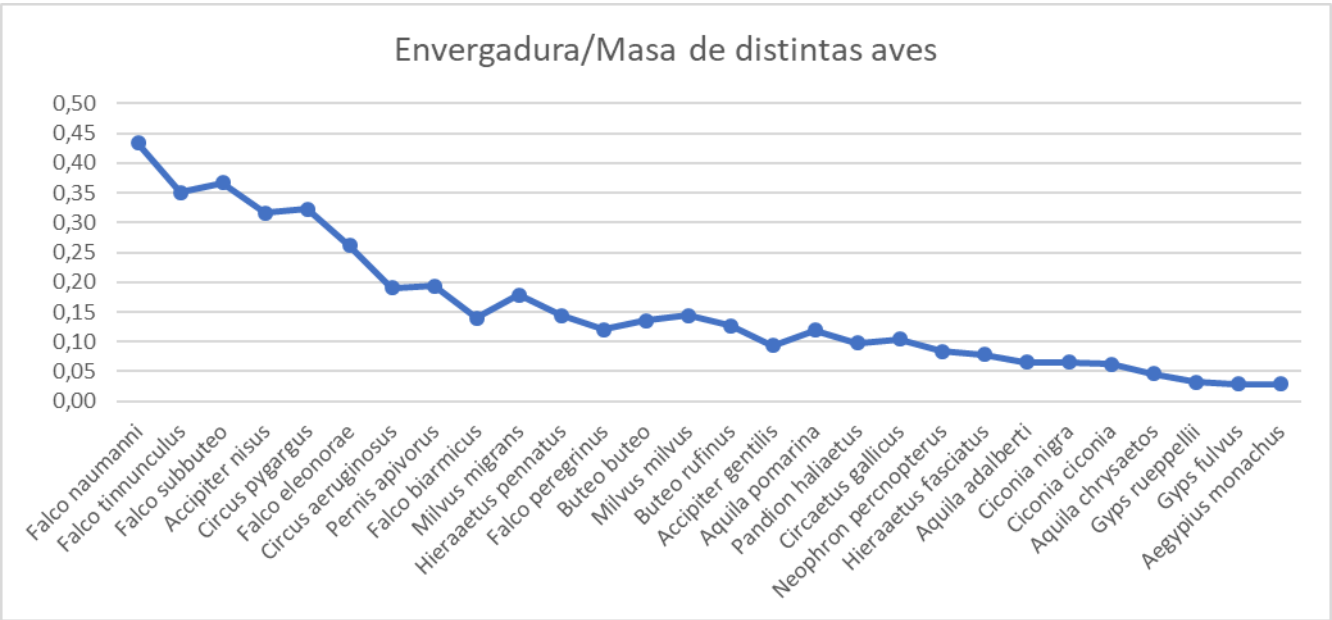
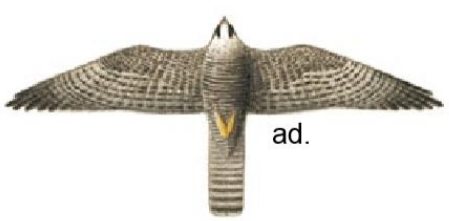


Ilustración 14:Comparación entre aves planeadoras migratorias del estrecho de Gibraltar.

Teniendo en cuenta los requisitos de peso y envergadura, los cuatro candidatos que más se adecúan a los criterios son: *Hieraaetus pennatus, Buteo buteo, Falco biarmicus* y *Falco peregrinus.* Podemos agrupar estas aves en dos grupos de geometrías similares: halcones (*Falco biarmicus, Falco peregrinus.*) y pequeñas águilas (*Hieraaetus pennatus, Buteo buteo*).



Falco peregrinus

Falco biarmicus

Buteo buteo

Hieraaetus pennatus

Ilustración 15: Comparación de la vista en planta de aves con las alas extendidas.

De las cuatro aves se decide estudiar el Águila Calzada (*Hieraaetus pennatus*), por tener la mejor relación entre envergadura y peso para los valores requeridos. Otras razones por las que se elige es que las águilas tienen un vuelo más calmado que los halcones, mayor superficie alar para la misma envergadura, es la más claramente migratoria y la más frecuente en la península.

El Águila Calzada (Hieraaetus pennatus) es el águila más pequeña de la Península ibérica, con una envergadura de 110-135 cm y un peso de 700-1000 g. Es un ave migratoria de largo alcance, con poblaciones ibéricas que migran en invierno al África Subsahariana. Como el resto de las águilas es capaz de mantenerse planeando a baja velocidad y baja altura haciendo apenas movimientos durante largos periodos de tiempo.

Se decide basar el diseño en un águila de unos 800 g de peso y 1,1 metros de envergadura.

6.2 Geometría del ala del RPAS

Se ha optado por una configuración de ala volante, resultado de eliminar el torso y la cola de la configuración alar del Águila Calzada. La configuración de ala volante permite reducir la resistencia, el peso y maximizar la sustentación, muy importante para una aeronave que busca planear con mucha autonomía. Como desventajas presentan problemas de estabilidad y control, pero para esta aeronave no son los parámetros que buscamos optimizar.

El resultado de eliminar el torso del ave es una aeronave de 1 metro de envergadura. El peso del ave se mantiene, aunque se retire el torso, por considerar que la configuración de ala volante podrá generar una sustentación igual a la del ave con torso, por lo que podrá planear con el mismo peso.

Ilustración 16: Águila Calzada.

Para conseguir una geometría parecida a la del águila se define la geometría del borde de ataque como una función polinómica y la del borde de salida como una función trigonométrica:

$$b.a. = -7 \cdot y^6$$

$$b.s. = -0.015 \cdot (1 + cos(2 \cdot \pi \cdot y))$$

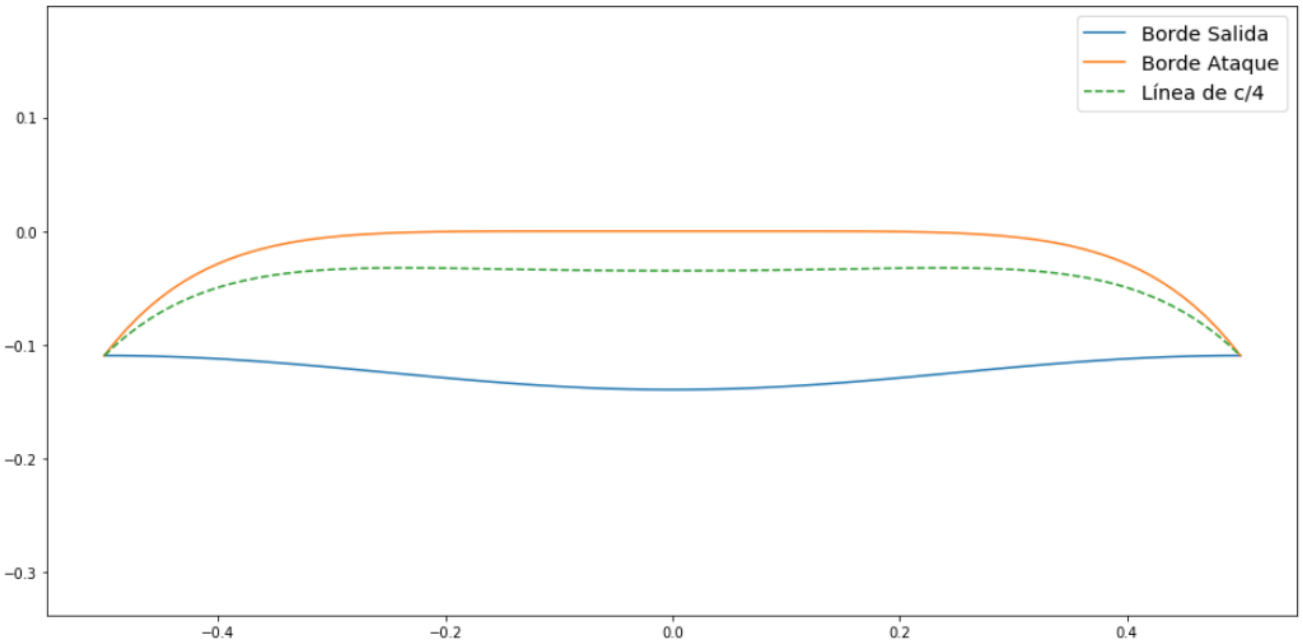


Ilustración 17: Vista en planta del RPAS.

Queremos añadir una articulación para emular la forma en la que las aves pliegan las alas mientras planean. Para ello se supone que el hueso coincide con la línea de c/4 y se añade una articulación en la mitad del semiala. La posición de estos huesos se controla con dos variables, una para la flecha de la sección central y otro para flecha de la sección de los extremos. El movimiento de las articulaciones es simétrico y las variables de control coinciden con la tangente a la línea c/4 en el punto de articulación.

El movimiento de las articulaciones se hace de forma que la superficie y el alargamiento efectivo se mantengan constantes.

S= 0.1087 m^2

A= 9.1954

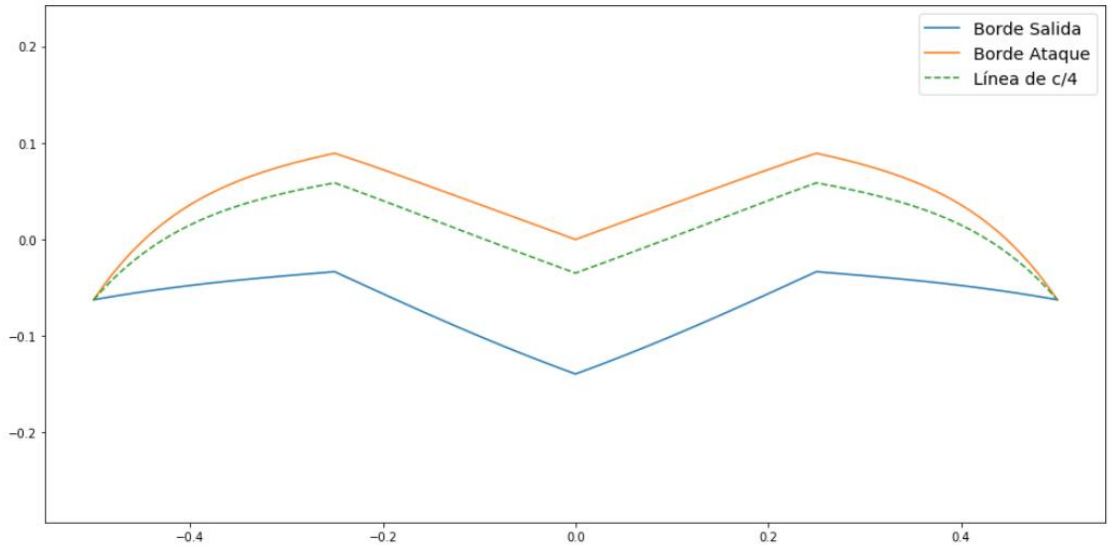


Ilustración 18: Articulación central -20º y articulación de los extremos 10º

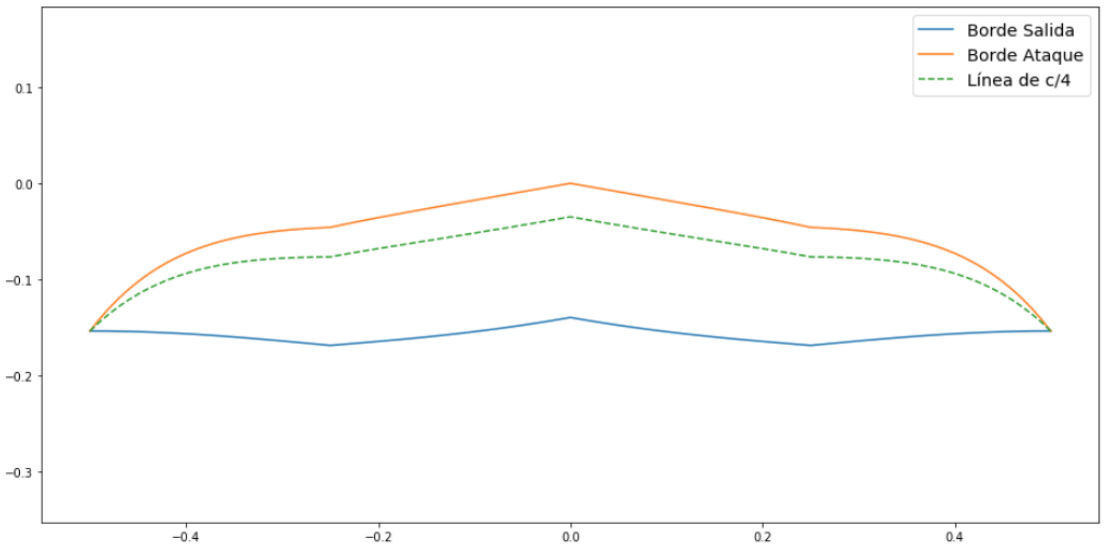


Ilustración 19: Articulación central 10º y articulación de los extremos 0º

Ilustración 20: Águila calzada con las alas flexionadas.

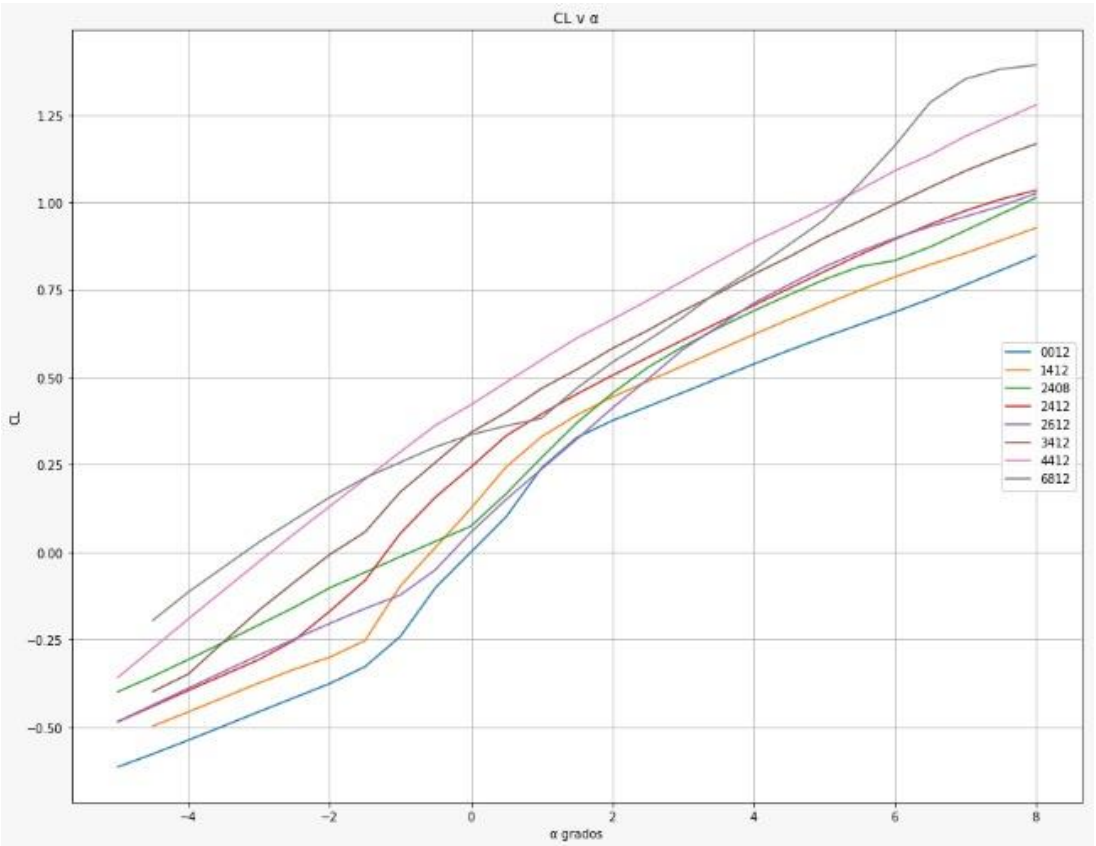
En el anexo 5 se puede encontrar descrito en detalle el código del programa que genera la geometría del ala volante.

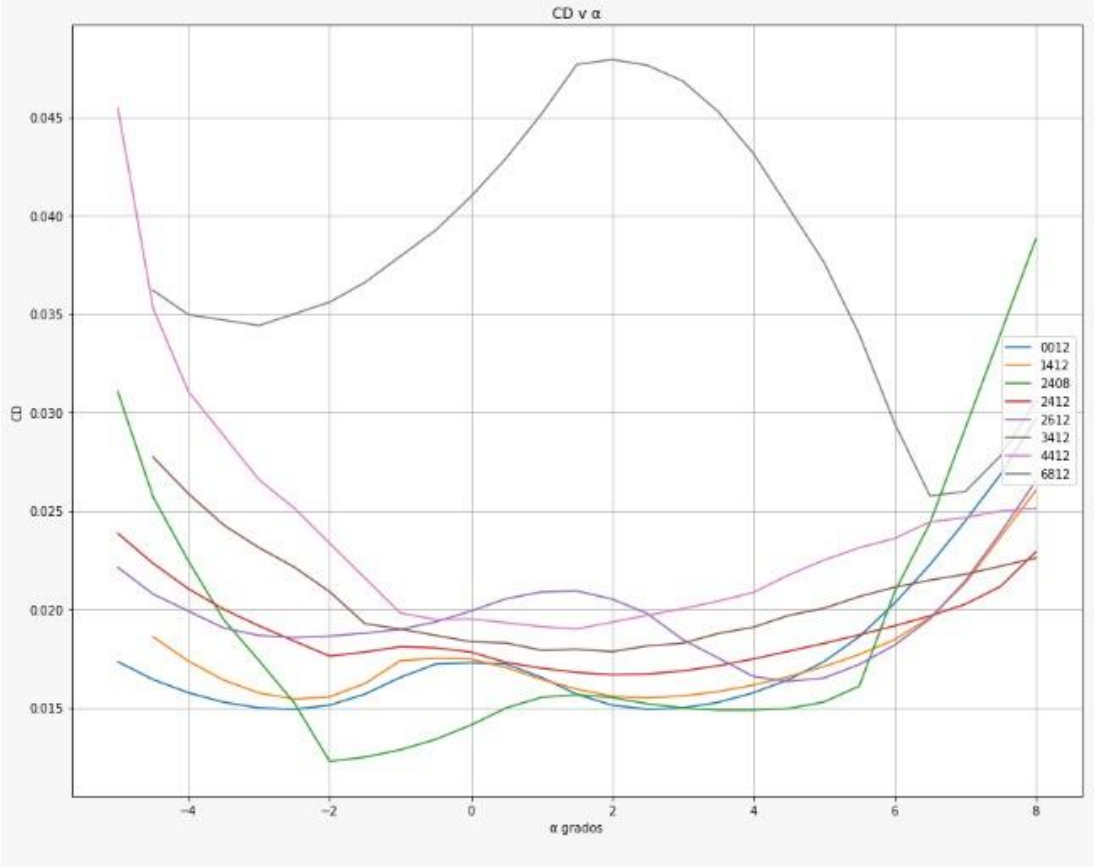
6.3 Elección del perfil alar.

Para hacer las simulaciones se elegirá una velocidad de 10 m/s que es una velocidad común de planeo para aves, presión de una atmósfera y temperatura de 20º. Por lo que Re=92658 y M=0.029

XFOIL permite trabajar con perfiles NACA de 4 y 5 cifras sin necesidad de introducir las coordenadas punto a punto. Se decide trabajar con perfiles NACA de 4 cifras para poder utilizar el código de representación 3-D.

Primero se comparan las características de los perfiles 2-D, se utiliza el modo viscoso de XFOIL, con un rango de ángulos de -4 a 8 grados.


CL v α
CL
α grados
0012
1412
2408
2412
2612
3412
4412
6812


CD v α
CD
α grados
0012
1412
2408
2412
2612
3412
4412
6812

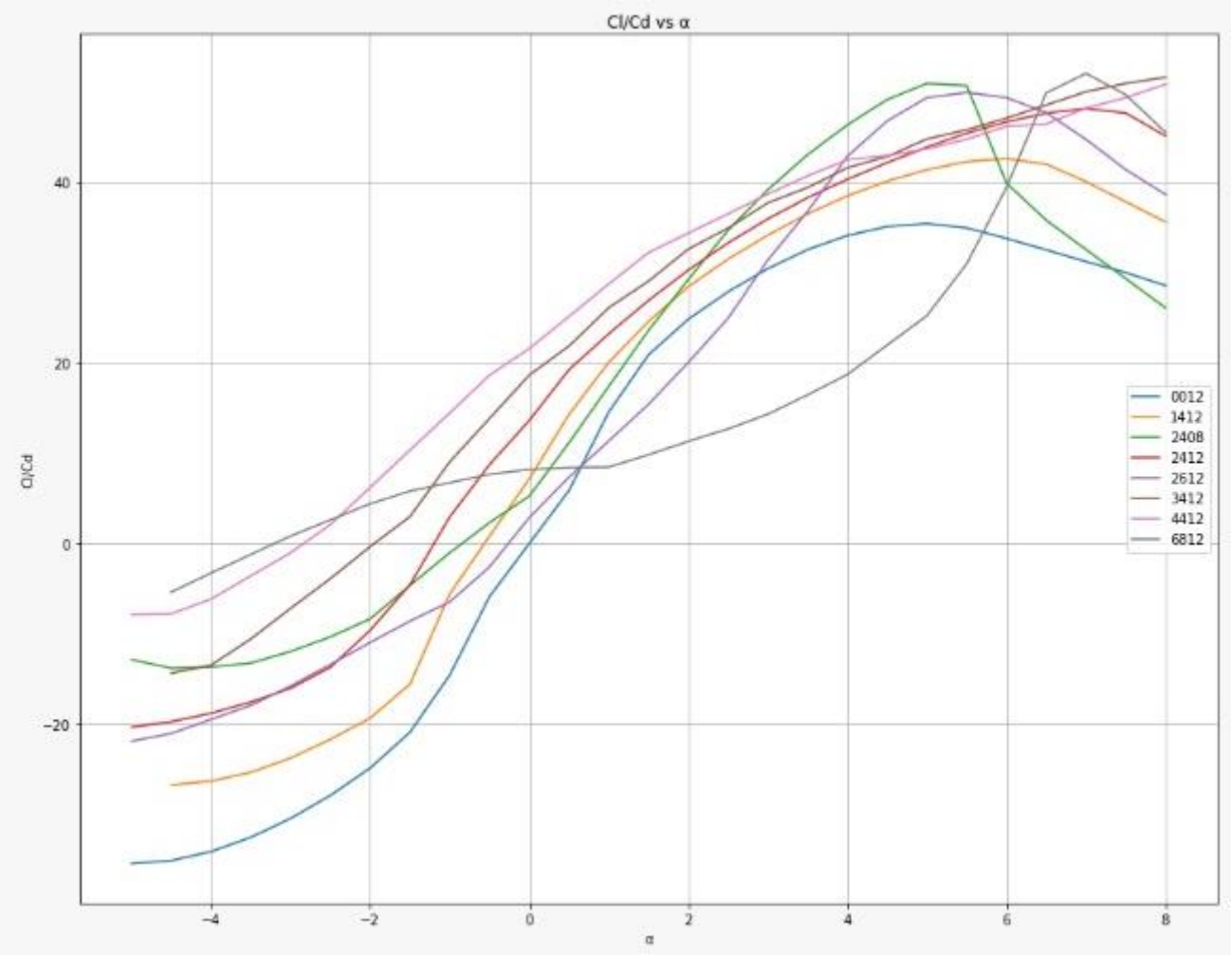


Ilustración 21: Resultados de la comparación de perfiles en XFOIL.

El modelo VLM que hemos utilizado linealiza la curva del coeficiente de sustentación por lo que se debe elegir un perfil con una zona lineal amplia para conseguir unos resultados lo más fiables posible. Para tener un planeador necesitamos un perfil con Cl/Cd elevado, poca resistencia y mucha sustentación. Teniendo en cuenta todas las opciones se elige el perfil NACA 4412, perfil hipersustentador de baja velocidad.

Se utiliza el código completo para verificar la elección de este perfil, en una configuración en la que ambas articulaciones están totalmente extendidas:

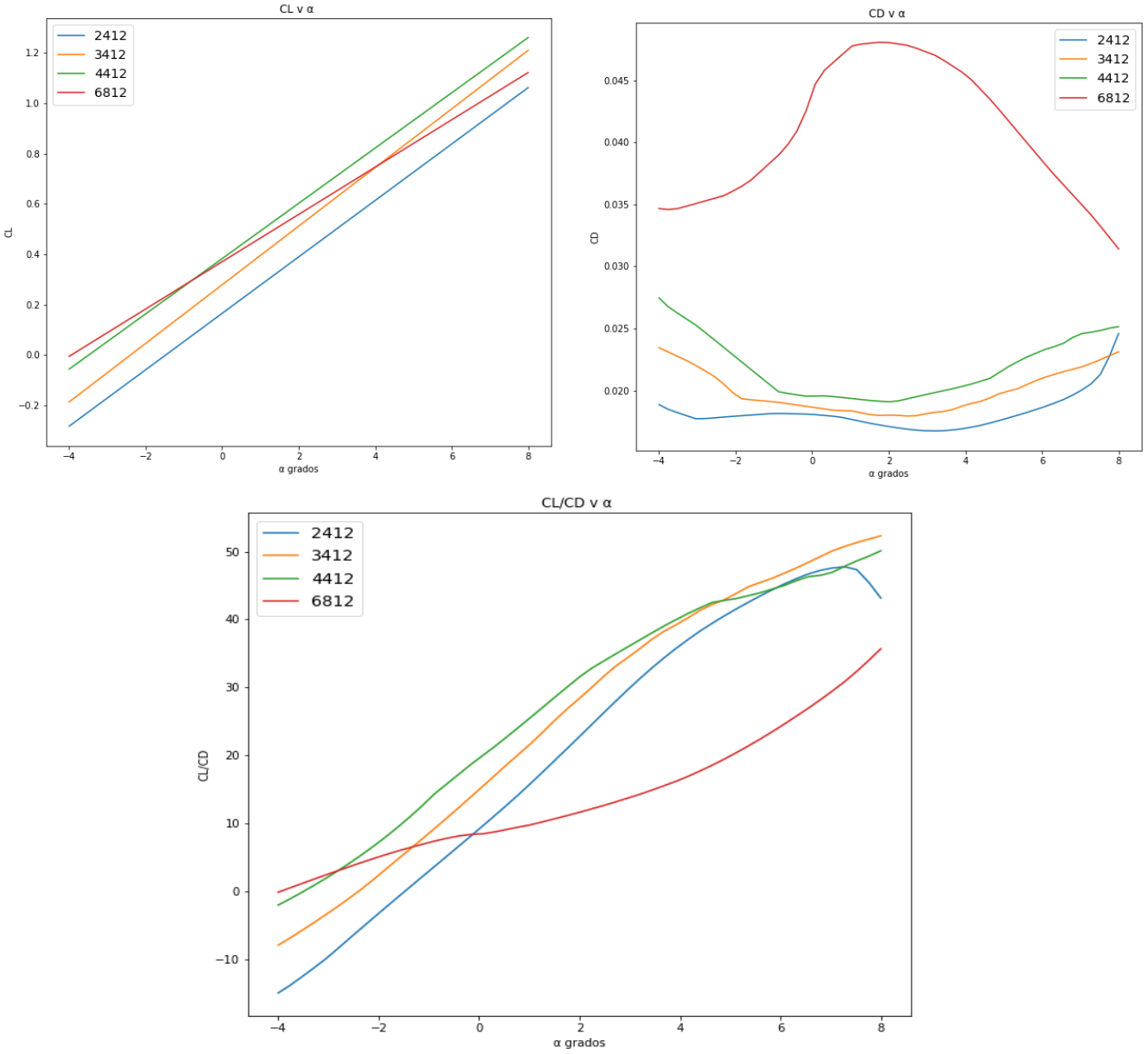


Ilustración 22: Resultados de la comparación de perfiles aplicada al ala.

Se puede verificar que la comparación es similar al caso del estudio de los perfiles 2-D. Por lo que elegimos el NACA 4412.

6.4 Elección de la posición de las articulaciones.

A continuación, vamos a estudiar el efecto que tiene el giro de las articulaciones en la polar. Primero se estudia el ala como si la articulación intermedia no existiera, es decir, toda el ala girada un mismo ángulo.

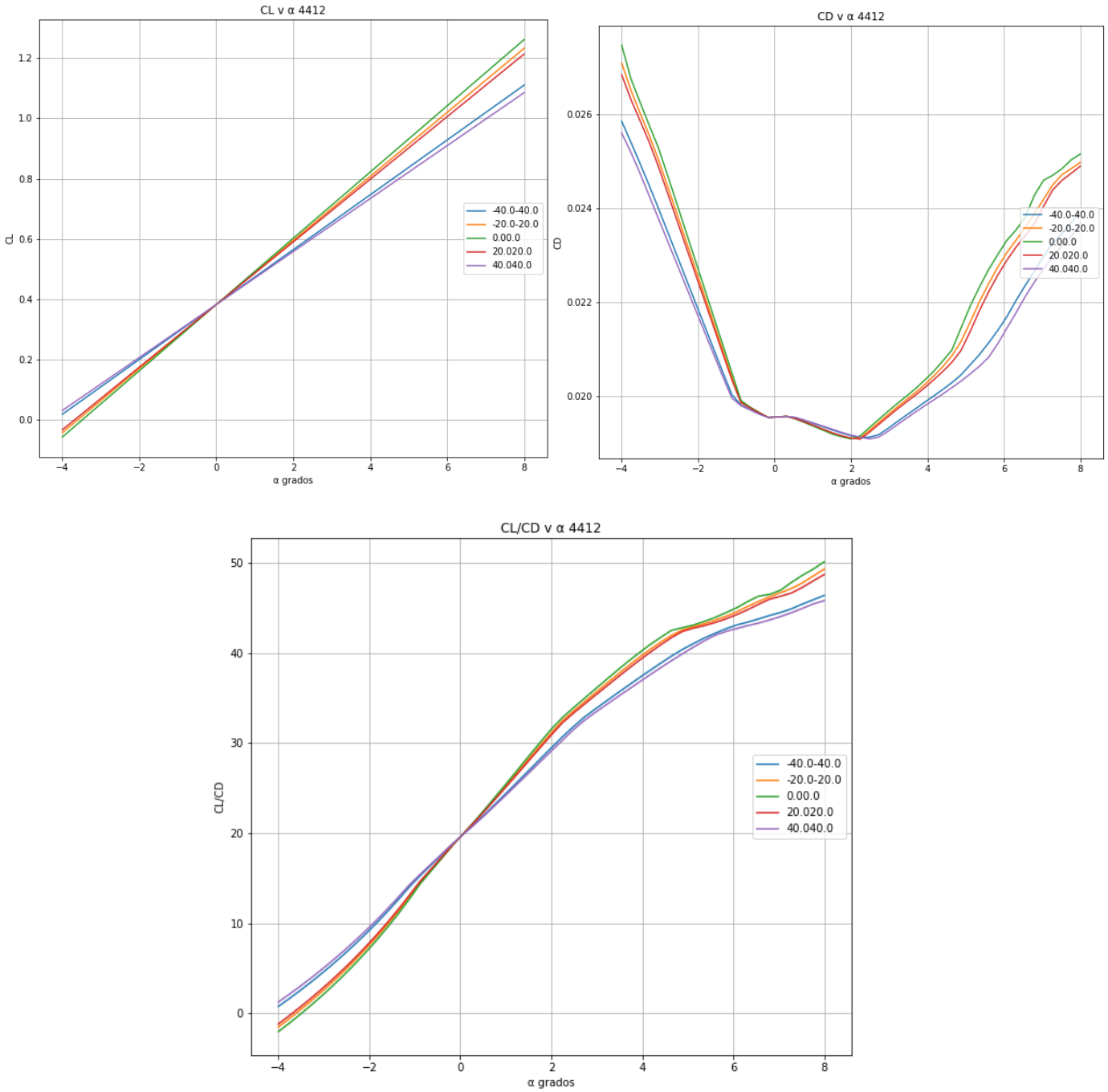


Ilustración 23: Comparación de giros de articulación, con los extremos fijos.

Se observa que la mejor configuración es la del ángulo igual a 0º. Las alas de aeronaves planeadoras no tienen flecha, por lo que este resultado es lógico.

A continuación, se estudia el efecto combinado del giro de ambas articulaciones, en las proximidades de 0º.

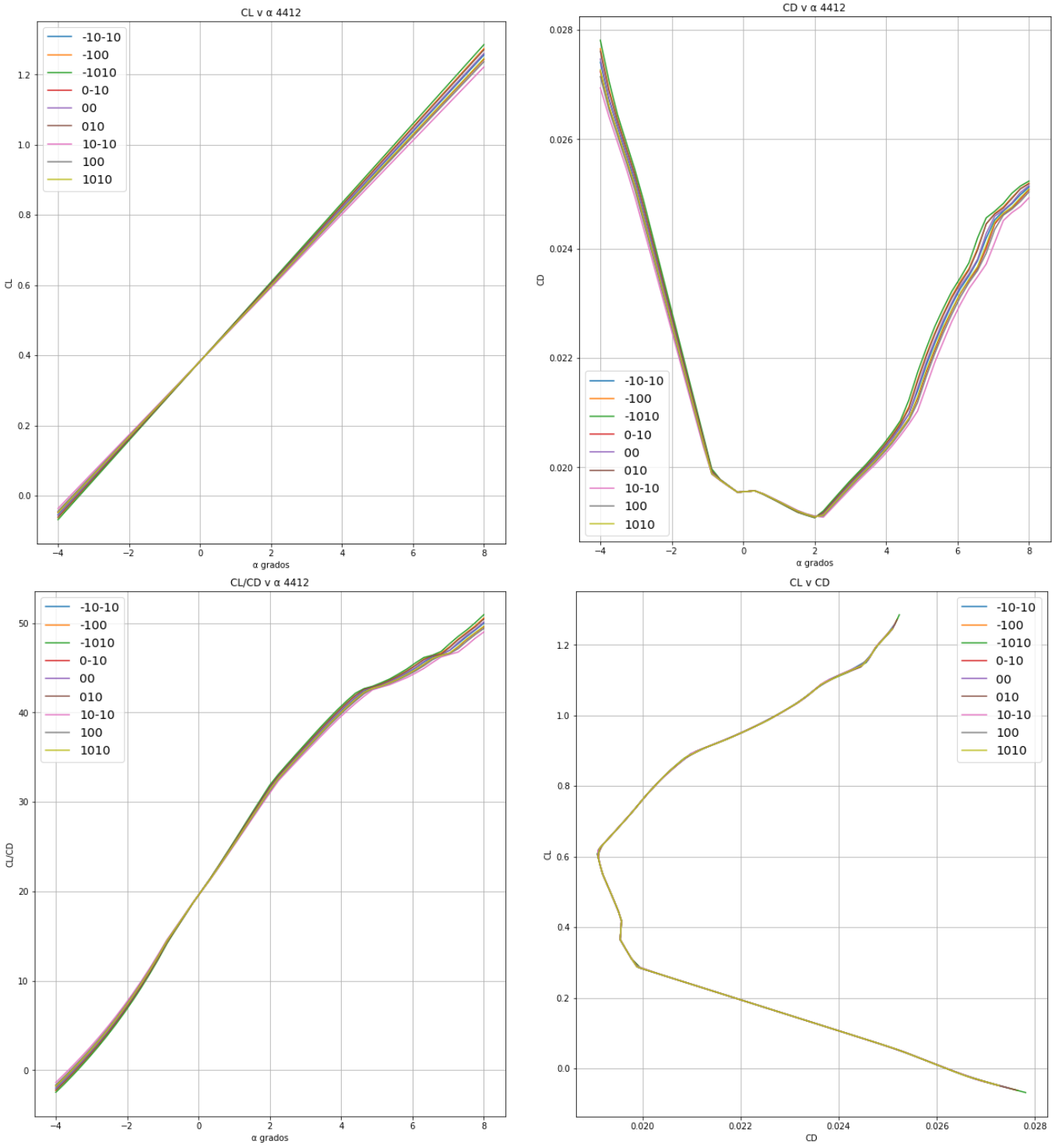


Ilustración 24: Comparación para distintos giros de articulaciones combinados.

Los valores son bastantes similares pero la configuración que consigue mejor relación CL/CD es la del ángulo de la sección central igual a -10º y el ángulo de las secciones laterales igual a 10º. Pequeñas variaciones de los ángulos tienen un efecto despreciable en la polar. Es una configuración que se puede observar en múltiples aves durante el vuelo en planeo como se puede ver en la siguiente imagen.

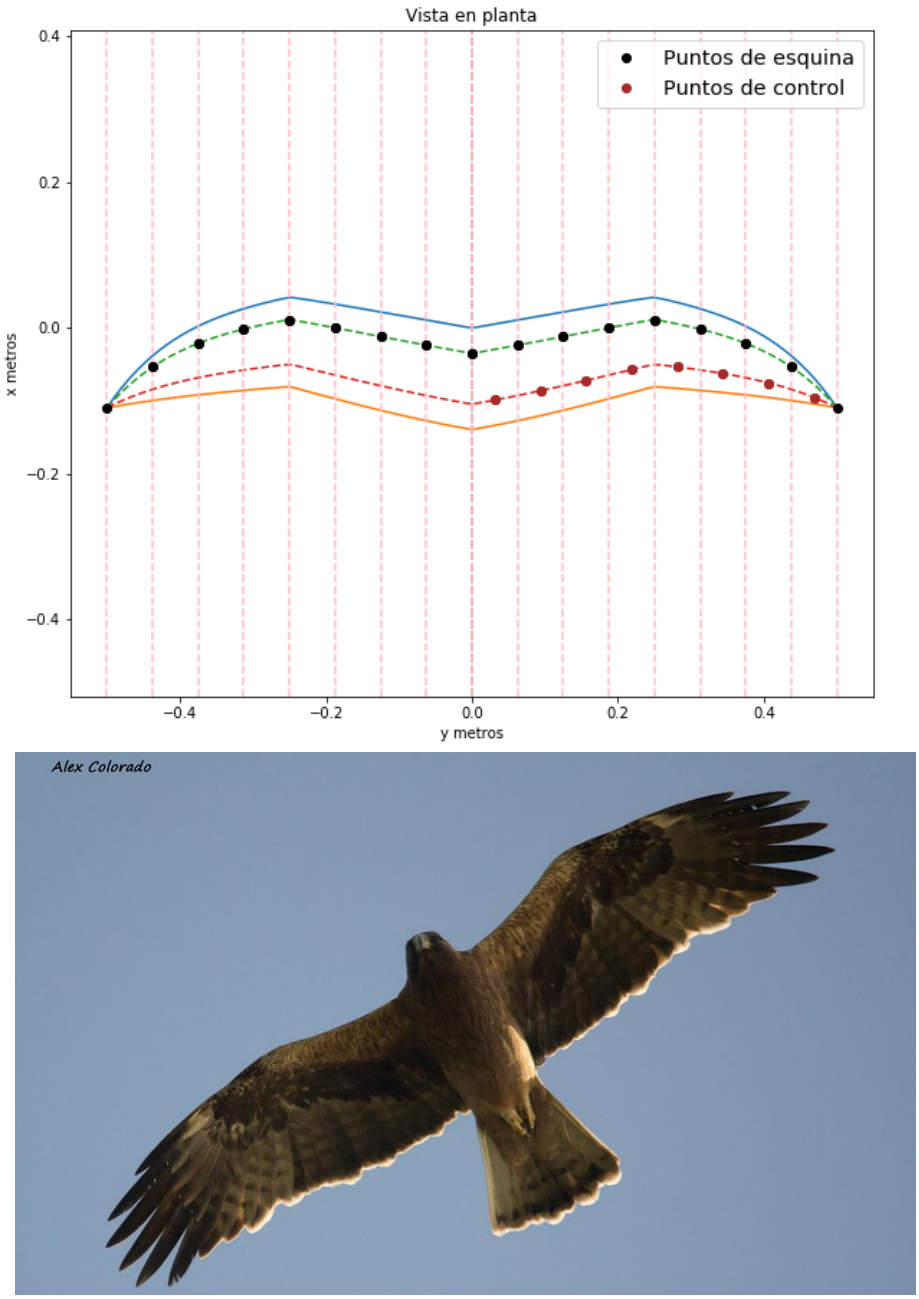


Ilustración 25: Comparación del ángulo de articulación obtenido y el observado en aves.

6.5 Efecto de la torsión.

Pese a que todos los cálculos se realizan sobre un modelo en 2-D del ala, se puede añadir el efecto de la torsión en la condición de contorno del VLM:

$$w_m \approx -V_\infty \cdot (\alpha + \varepsilon(y))$$

Se modeliza la torsión de forma lineal desde el centro del ala hasta la punta. Por no tener fuselaje la torsión en la base del ala es como un cambio del ángulo de ataque, por ello solo se va a trabajar con el cambio de torsión en la punta. El efecto que tiene en la polar es:

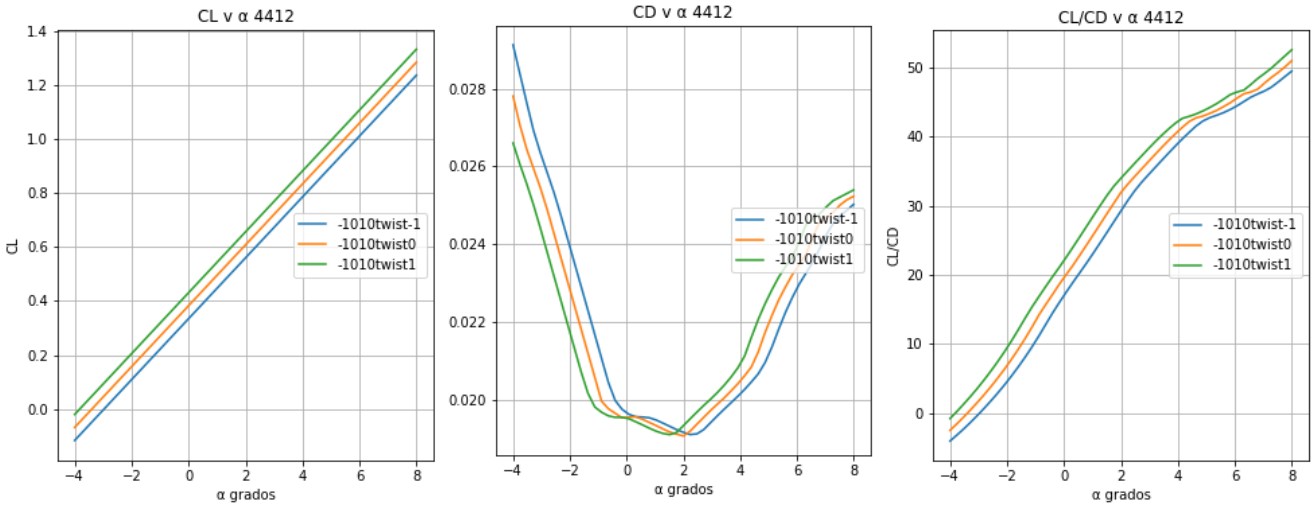


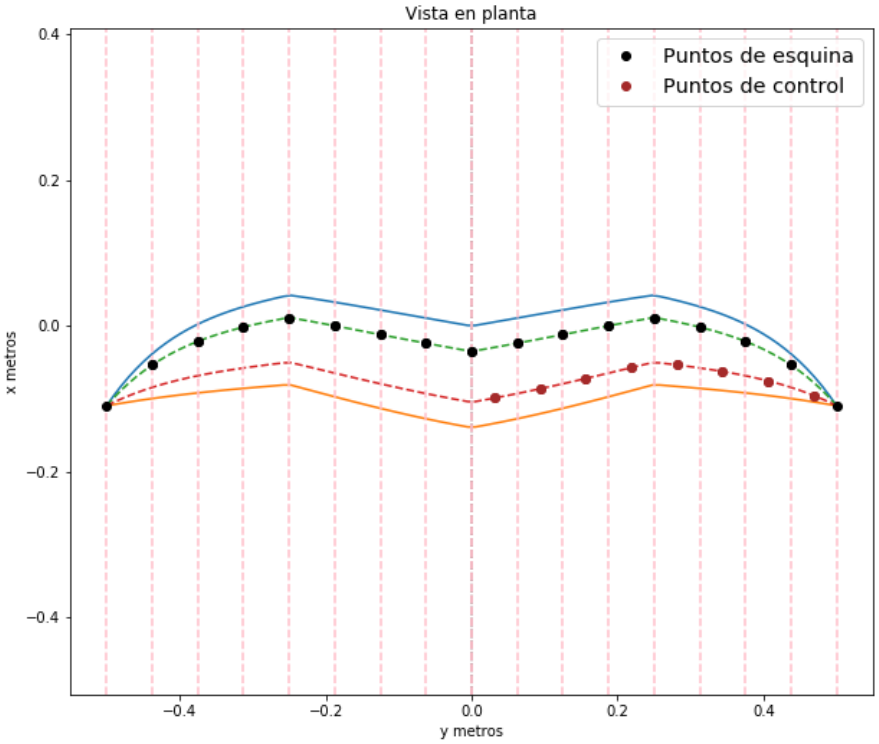
Ilustración 26: Efecto de la torsión.

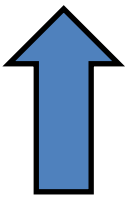
Se puede observar que el efecto de la torsión es un desplazamiento de las curvas hacia la izquierda para torsiones crecientes. Para el caso de estudio se va a mantener la torsión igual a cero. Se puede observar en las águilas que el cambio de torsión de un semiala puede utilizarse como control, convirtiendo el ala en asimétrica.

6.6 Configuración elegida.

Para resumir la configuración del ala elegida:
Ala volante de 1 metro de envergadura, 800g de peso, basado en el ala del águila calzada, sin torsión y con las articulaciones a -10º y 10º. Al ser un diseño preliminar no se eligen superficies verticales, aunque en un estudio más en detalle se debería añadir alguna superficie vertical al ala.

Vista en planta:



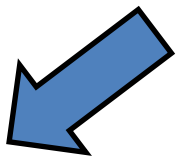


Sentido de avance.

Ilustración 27: Vista en planta de la configuración elegida.

Representación 3D del RPAS:

Sentido de avance.



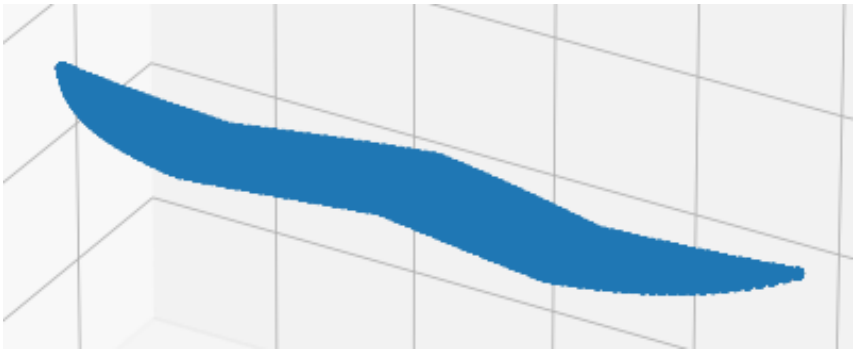


Ilustración 28: Representación 3-D del RPAS con las articulaciones a -10 y 10 grados.

Coeficientes aerodinámicos de la configuración definitiva:

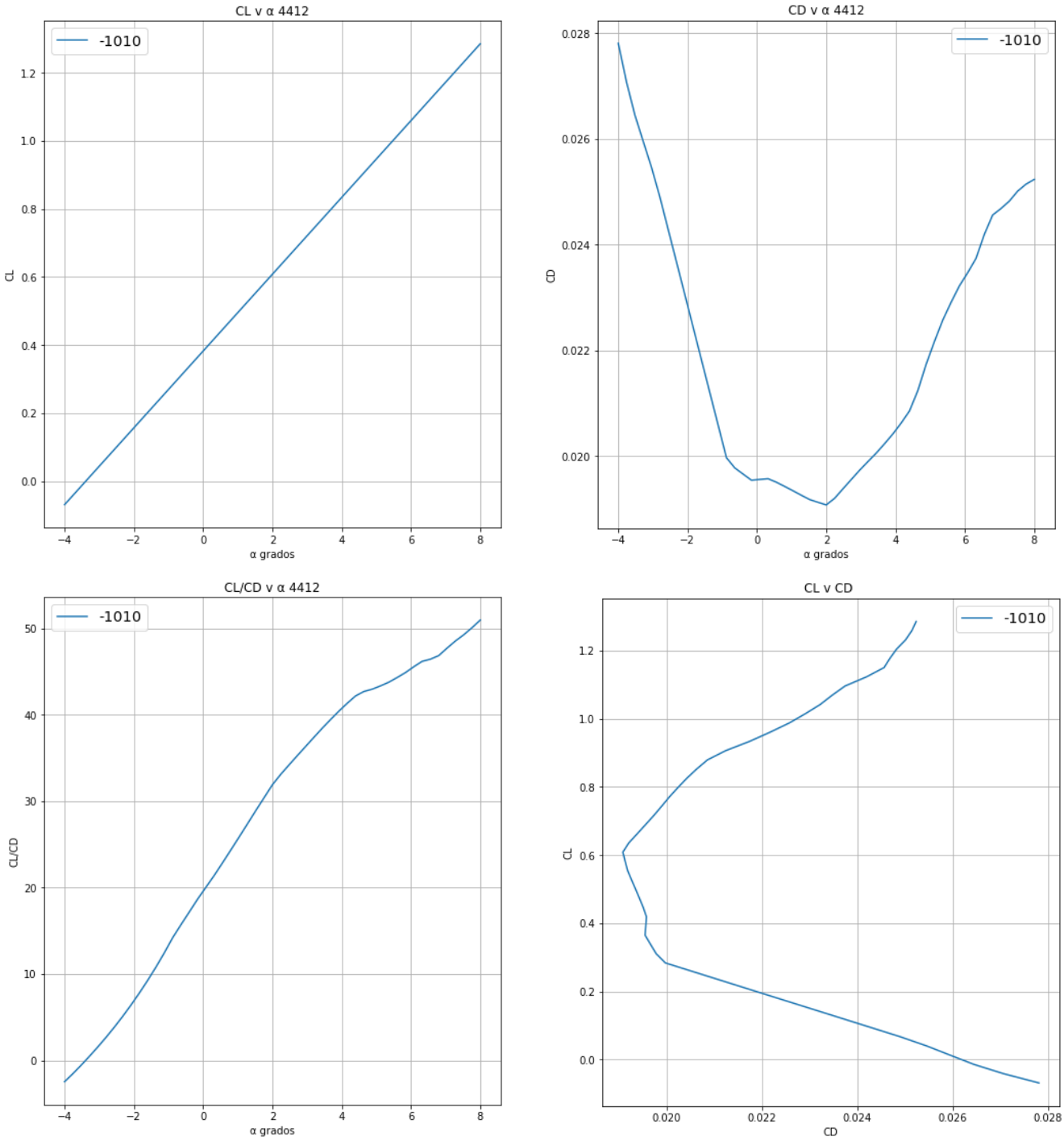


Ilustración 29:Coeficientes de sustentación y resistencia de la configuración elegida.

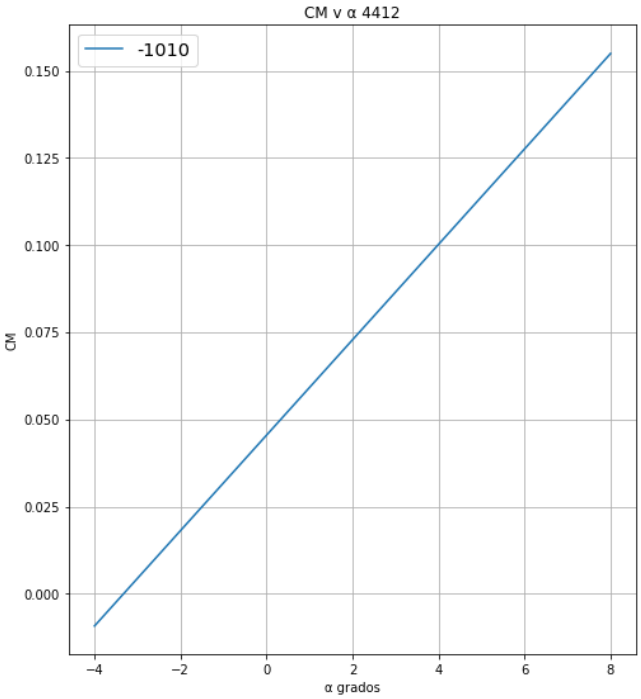


Ilustración 30: Coeficiente del momento de cabeceo de la configuración elegida.

7 Planta de potencia.

Teniendo en cuenta la información obtenida del estudio de aeronaves semejantes se elige un motor eléctrico de 20W de potencia. Es un motor más que suficiente para que esta aeronave realice un vuelo horizontal a velocidades cercanas a 10 m/s, pero tiene potencia extra para poder despegar y realizar otras maniobras.

7.1 Elección de la hélice.

Se pretende encontrar una hélice comercial existente para poder determinar las actuaciones de la aeronave correctamente. Se elegirá una hélice del catálogo de APC Propellers, para ello se buscará la que tenga el mejor rendimiento propulsivo máximo.

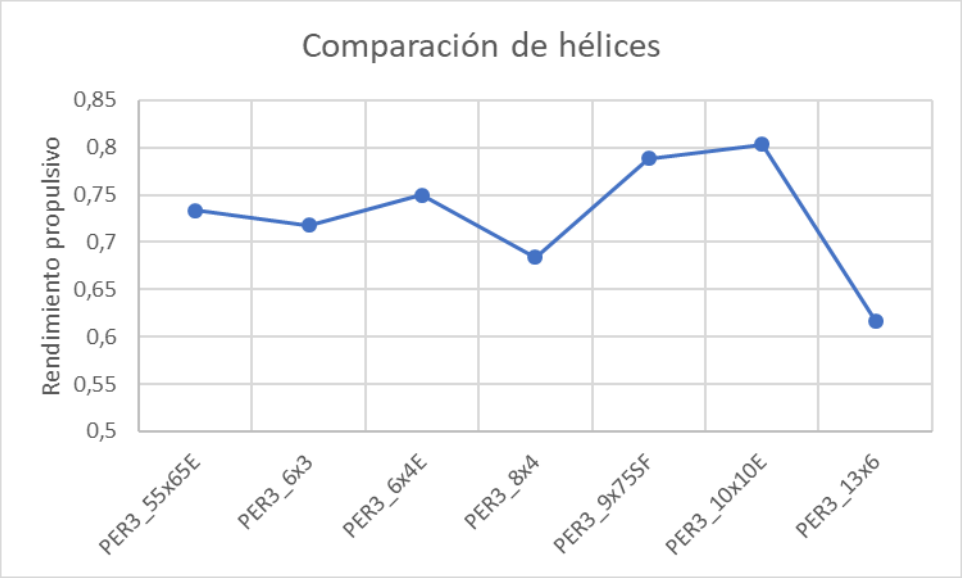


Ilustración 31: Comparación del rendimiento de distintas hélices.

Por lo que se elige la PER3 10x10E. A continuación, se estudia la evolución del rendimiento propulsivo con la velocidad a distintas revoluciones del motor.

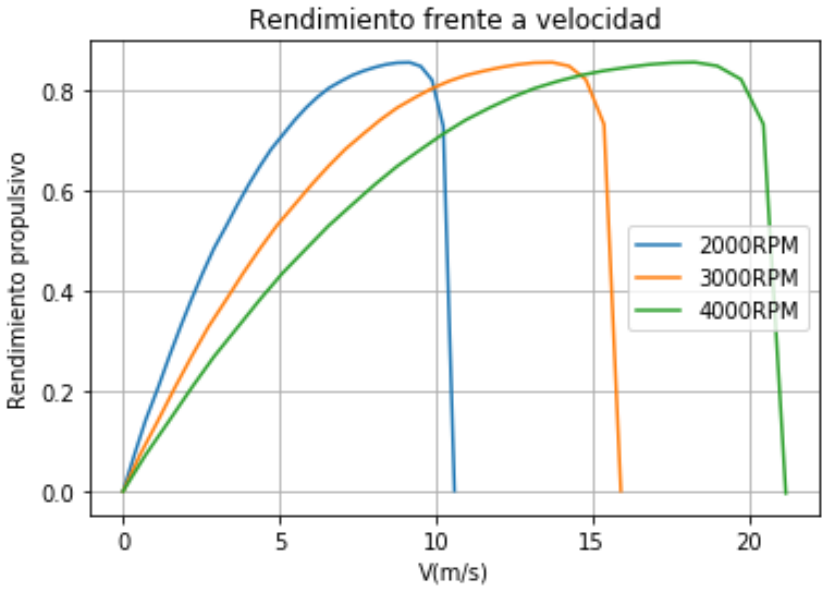


Ilustración 32: Rendimiento propulsivo de la hélice.

Se puede observar que el rendimiento propulsivo máximo se mantiene constante, pero varía la velocidad de avance a la que se obtiene.

Para dar cierto margen al vuelo ascendente se decide utilizar la hélice PER3 10x10E a 3000RPM, por lo que la potencia real disponible en función de la velocidad es:

$$Pot\ disp = Pot\ motor \cdot \eta$$

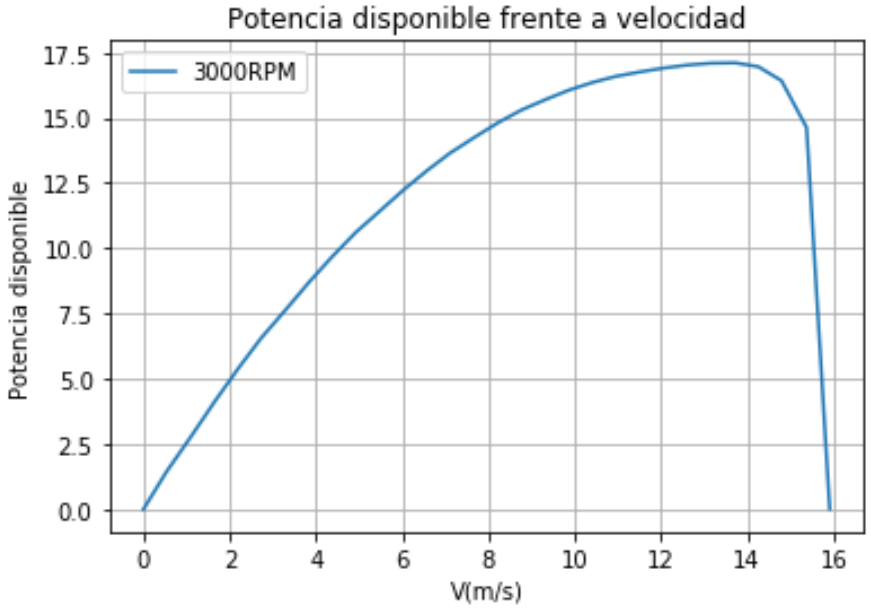


Ilustración 33: Potencia disponible frente a velocidad.

8 Actuaciones.

Las actuaciones se ocupan del estudio del movimiento del centro de masas del avión a lo largo de su trayectoria. Su propósito es establecer la configuración del avión más adecuada para realizar una determinada misión o definir la utilidad operacional de un avión ya diseñado.

Las ecuaciones generales que rigen el vuelo de una aeronave son:

$$-mg\,\sin(\gamma) + T\cos(\varepsilon)\cos(\nu) - D = mV$$
$$mg\,\sin(\mu)\cos(\gamma) + T\cos(\varepsilon)\sin(\nu) - Q = mr_w V$$
$$mg\,\cos(\mu)\cos(\gamma) - T\sin(\varepsilon) - L = -mq_w V$$

Se toman las siguientes hipótesis:

- La aeronave se comporta como un sólido rígido. Las deformaciones que sufre no alteran las características aerodinámicas y no resulta necesario su estudio.
- La aeronave es simétrica.
- La aeronave se comporta como un cuerpo de masa puntual. Esto es debido a que no se consideran los grados de libertad de rotación que posee y se supone que durante todo el vuelo las superficies aerodinámicas proporcionan el equilibrio en este aspecto.
- La gravedad es constante e igual a 9.81 m/s^2.

8.1 Vuelo horizontal

Para una aeronave en vuelo horizontal, rectilíneo y uniforme propulsada por hélice se tienen las siguientes ecuaciones:

$$P_R = T_R \cdot V = D \cdot V$$
$$L = W$$
$$D = \frac{1}{2} \cdot \rho \cdot V^2 \cdot S \cdot C_D$$
$$L = \frac{1}{2} \cdot \rho \cdot V^2 \cdot S \cdot C_L$$

$$C_L = \frac{W}{\frac{1}{2} \cdot \rho \cdot V^2 \cdot S}$$

$$P_R = \frac{1}{2} \cdot \rho \cdot V^3 \cdot S \cdot C_D(C_L)$$

Para resolver primero determinamos el coeficiente de sustentación necesario, a nivel de mar y con 800g de peso. Luego utilizamos la polar obtenida del estudio aerodinámico para calcular potencia necesaria. El resultado se muestra a continuación:

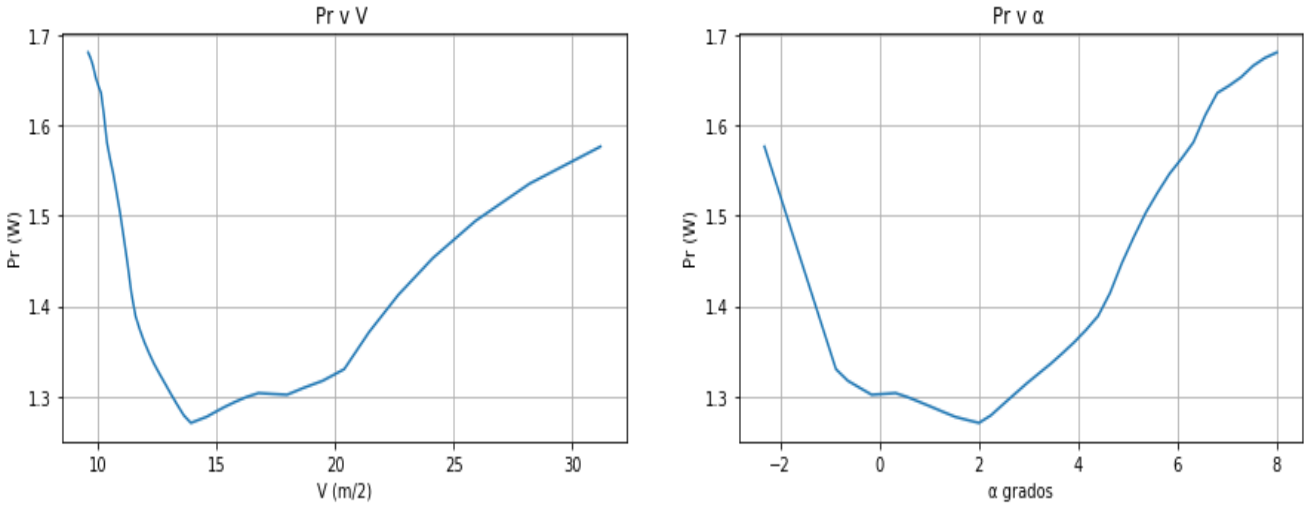


Ilustración 34: Potencia requerida a distintas velocidades.

Con una potencia mínima P_R=1.2W para 14m/s y ángulo de ataque de 2º al nivel del mar.

8.2 Planeo

En la condición de planeo para la aeronave se tienen las siguientes ecuaciones:

$$W = m \cdot g$$

$$W \cdot \sin(\gamma_D) = D$$

$$W \cdot \cos(\gamma_D) = L$$

Por lo que el ángulo de descenso equilibrado de la aeronave y la velocidad de descenso equilibrado son respectivamente:

$$\tan(\gamma_D) = \frac{1}{C_L/C_D}$$

$$V = \sqrt{\frac{2 \cdot \cos(\gamma_D)}{\rho \cdot C_L} \cdot \frac{W}{S}}$$

La velocidad de descenso es:

$$V_D = \frac{C_D}{C_L^{3/2}} \sqrt{\frac{2 \cdot W}{\rho \cdot S}}$$

Para evitar singularidades se evitará estudiar el planeo en la zona de $C_L \leq 0$.

Por lo que para el RPAS de estudio:

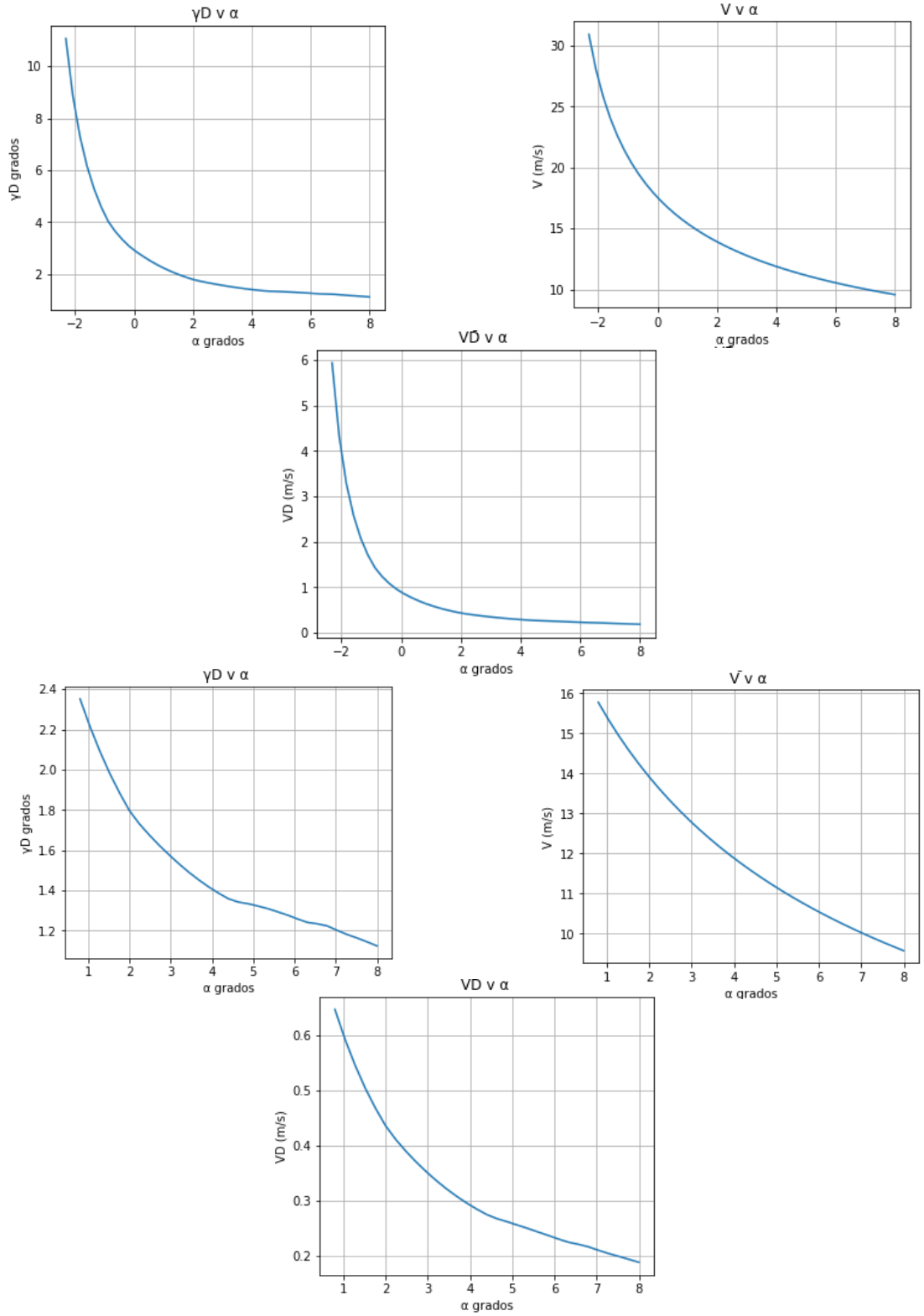


Ilustración 35: Parámetros de planeo del RPAS a distintos rangos de ángulos.

Podemos observar que la velocidad de planeo del águila, 10 m/s, se puede obtener entre 7 y 8 grados y una velocidad de descenso de 0.22 m/s.

8.3 Vuelo estacionario en subida.

La velocidad ascensional se puede calcular como la diferencia entre la potencia disponible y la necesaria para esa velocidad en relación con el peso de la aeronave:

$$V_S = \frac{P_{disp} - P_{nec}}{W} = \frac{P_{motor} \cdot \eta - P_{nec}}{W} = \frac{Exceso\ de\ potencia}{W}$$

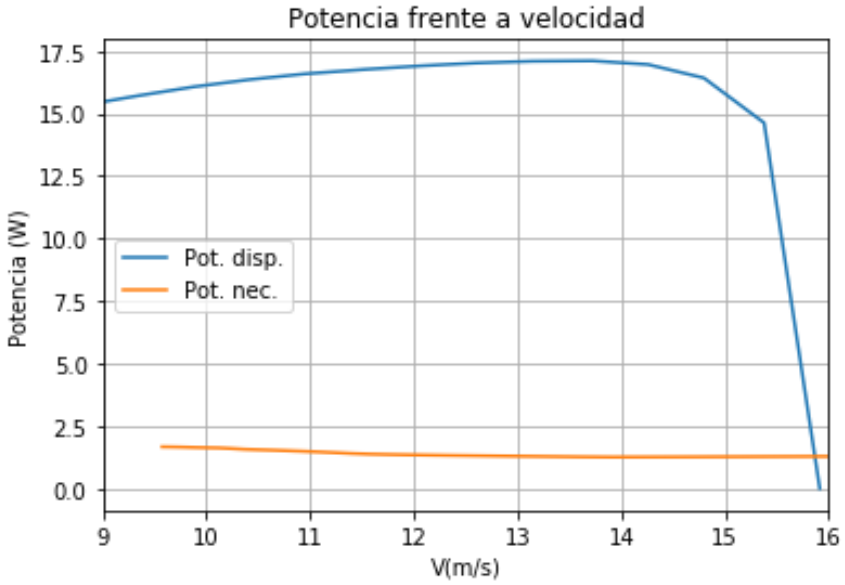


Ilustración 36:Potencia necesaria y potencia disponible frente a velocidad.

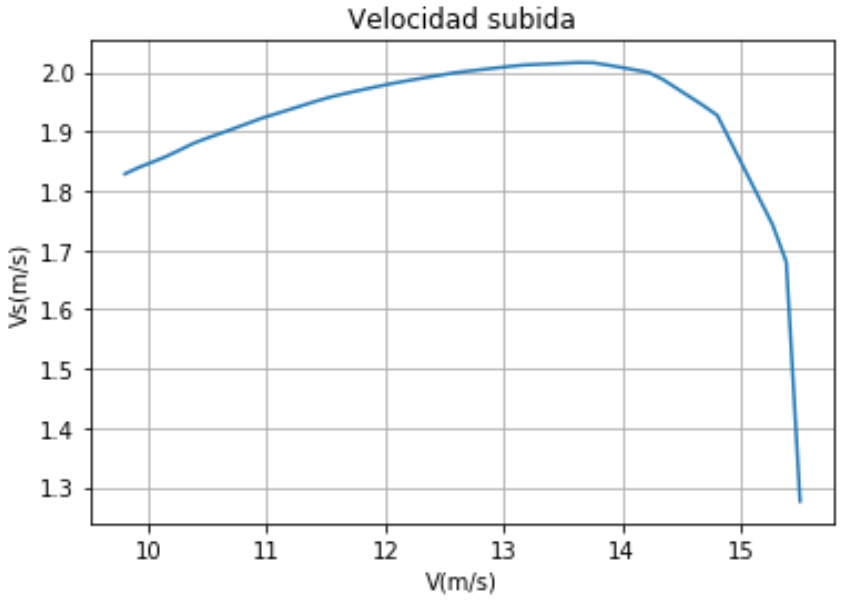


Ilustración 37: Velocidad de subida frente a velocidad de vuelo.

El punto donde la velocidad ascensional es máxima se obtiene cuando la diferencia entre la potencia disponible y la necesaria es máxima. Esta velocidad ascensional es suficiente para el despegue y adquirir la altura necesaria de la operación. Aun así, por su pequeño tamaño, se pueden utilizar otros métodos habituales para el lanzamiento de RPAS como pueden ser: catapultas, lanzamiento manual o remolque hi-start con cuerda elástica.

8.4 Velocidad de entrada en pérdida.

Para comprobar que el vuelo a 10 m/s es posible se calcula la velocidad de entrada en pérdida a varias alturas. Tomaremos la altura máxima de vuelo como el techo de nubes, 450m. Para el coeficiente de sustentación del ala se utilizará la aproximación:

$$CLmax = 0.9 \cdot Clmax * \cos(\Lambda) = 1.295$$

$$V_{STALL} = \sqrt{\frac{2}{\rho} \cdot \frac{W}{S} \cdot \frac{1}{(C_L)_{MAX}}}$$

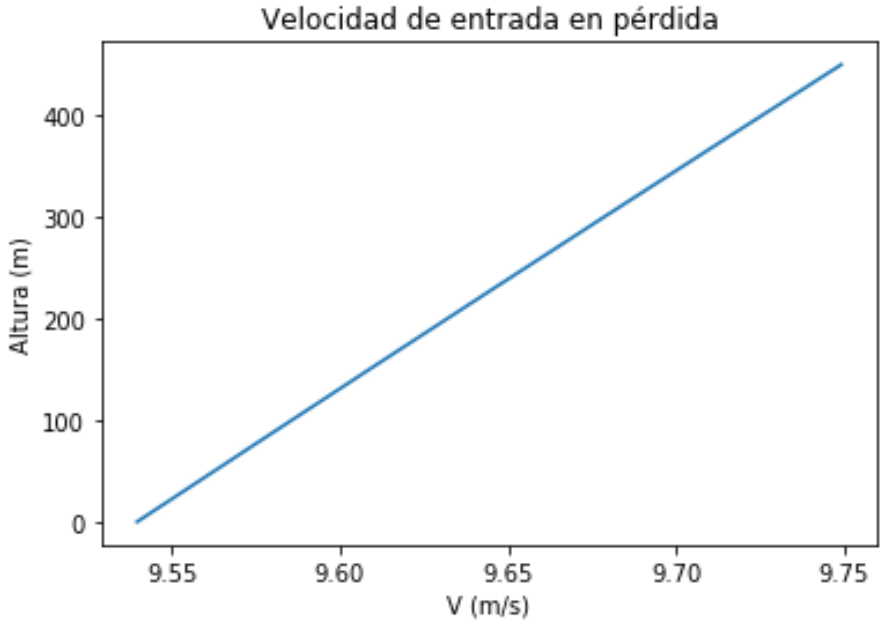


Ilustración 38: Velocidad de pérdida.

Como la máxima altura de vuelo es baja, la variación de la densidad es pequeña y la variación de la velocidad de perdida es mínima. En cualquiera de las alturas la velocidad de entrada en pérdida es menor a 10 m/s, por lo que los resultados son satisfactorios.

8.5 Velocidad máxima horizontal.

De la gráfica de potencia disponible y potencia necesaria podemos obtener la velocidad máxima horizontal que corresponde con el punto de intersección.

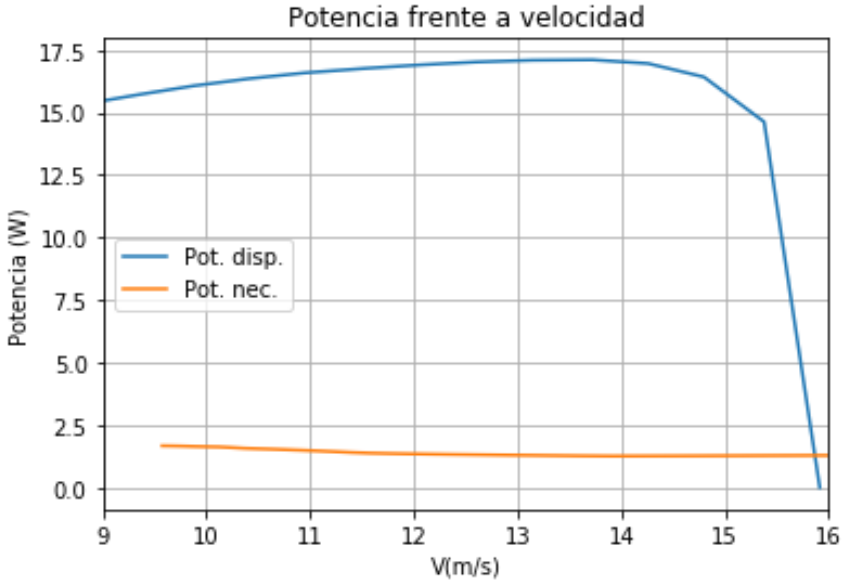


Ilustración 39: Potencia necesaria y potencia disponible.

La velocidad máxima a nivel del mar son 15.87 m/s y la variación hasta el techo de vuelo es insignificante. Aumentando las RPM del motor se pueden conseguir velocidades máximas superiores.

8.6 Envolvente de vuelo.

La envolvente de vuelo está limitada por el techo de nubes, la velocidad de entrada en pérdida y la velocidad de vuelo máxima horizontal.

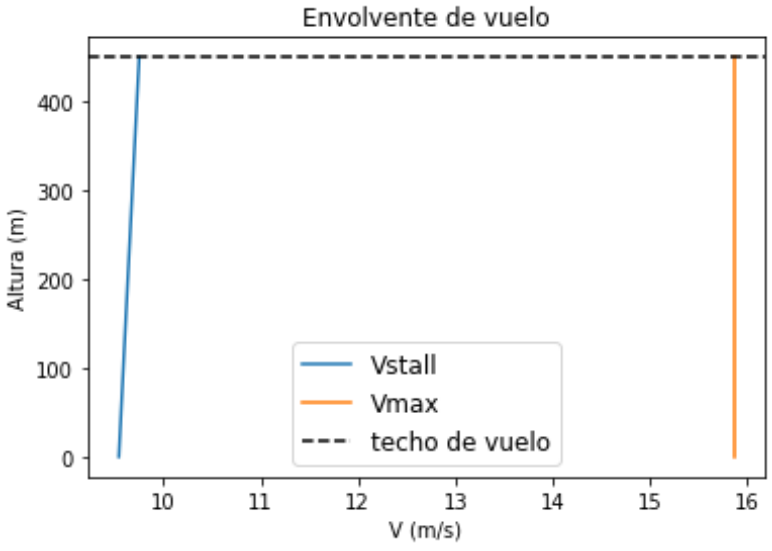


Ilustración 40: Envolvente de vuelo.

9 Estabilidad

La estabilidad se define como la capacidad que tiene un avión a regresar a la condición de equilibrio de forma autónoma. Se distingue entre estabilidad estática y dinámica:

- La estabilidad estática es la tendencia del avión a generar momentos, frente a una perturbación, que hagan que el avión vuelva a la posición de equilibrio de partida.
- La estabilidad dinámica mide la tendencia del avión a recuperarse de una perturbación a lo largo del tiempo.

Para que la aeronave sea dinámicamente estable es necesario que sea estáticamente estable. Sin embargo, no se trata de una doble implicación, por lo que una aeronave estáticamente estable puede resultar inestable dinámicamente. En este documento nos vamos a centrar en el cálculo de la estabilidad estática.

9.1 Estabilidad estática longitudinal.

La estabilidad estática se caracteriza por el margen estático con mandos fijos (H_0), que representa la distancia adimensional entra la posición del centro de masas del avión y el punto neutro con mandos fijos $(\hat{N}_0)$:

$$H_0 = \hat{N}_0 - \hat{x}_{CG}$$

El centro de gravedad se obtiene según la posición y masa de los distintos componentes:

	Masa (g)	Xcg (m)
Carga de pago	172	0,03
Raspberry	60	0,04
Servo (x2)	22	0,06
Receptor	10	0,02
ESC	36	0,01
Motor	50	0,12
Batería	150	0,08
Cuerpo	300	0,06
Total	800	0,0568

Se calcula como:

$$x_{cg} = \frac{\sum_i x_i \cdot m_i}{\sum_i m_i} = 0.057m$$

El punto neutro con mando fijo es la posición del centro de masas del avión (adimensionalizado con la cuerda media del ala) que anula $C_{M.}$ y se define como:

:

$$N_0 = \hat{x}_{acwb} + \frac{a_t}{a_{wb}} \eta_t \cdot \hat{V}_t \left(1 - \frac{\partial \varepsilon}{\partial \alpha}\right)$$

Al tratarse de un ala volante el segundo sumando es cero. Por tanto, el punto neutro con mando fijo será la distancia a la cual el C_M se iguala a cero:

$$N_0 = \hat{x}_{CG}|_{CM=0}$$

Sustituyendo en la ecuación del coeficiente de momentos:

$$C_M = C_{M_{OY}} - C_L\left(\widehat{x_{cg}} - \widehat{N_0}\right)$$

$$N_0 = \widehat{x_{cg}} - \frac{C_{M_{OY}}}{C_L}$$

$$H_0 = \widehat{N_0} - \widehat{x_{cg}}$$

El punto neutro con mando fijo y el margen estático con mandos fijos varían con el ángulo de ataque como se puede observar en la siguiente gráfica:

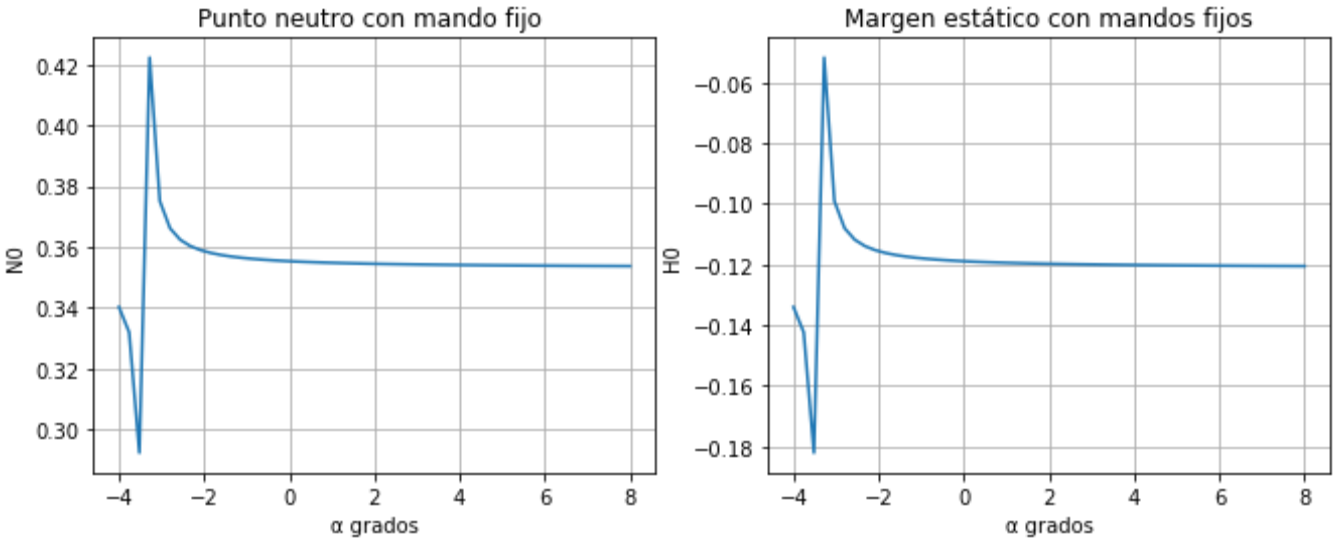


Ilustración 41: Punto neutro y margen estático con mandos fijos.

De lo que se deduce que la aeronave es inestable. En un planeador buscamos que sea estable por lo que se deberá añadir una superficie de cola, modificar el centro de gravedad o utilizar un sistema electrónico para conseguirlo.

10 Conclusiones.

Podemos utilizar la geometría alar del águila calzada para diseñar un micro RPAS capaz de ser utilizado en enjambre para tareas de vigilancia. Este acercamiento de replicar un ave para diseñar una aeronave se puede aplicar a otras misiones.

El código descrito en este TFG permite realizar el diseño previo de una aeronave de forma rápida y precisa. Es fácilmente adaptable a cualquier geometría alar y permite variar de forma inmediata múltiples parámetros para comparar y tomar decisiones de diseño.

Bibliografía.

- *Airfoiltools.com*
- Miguel A. Barcala Montejano y Angel A. Rodriguez Sevillano. *Diseño Preliminar de un RPAS.* Madrid, 2015.
- Miguel Ángel Gómez Tierno, Manuel Pérez Cortés y César Puentes Márquez. *Mecánica del Vuelo.* 2ª Edición. Garceta.
- John J. Bertin y Michael L. Smith. *Aerodynamics for engineers.* 2ª Edición.
- Emilio Pérez Cobo. *Apuntes de Cálculo de Aviones.* ETSIAE, 2015.
- David Barros Cardona y David Ríos Esteban. *Guía de aves del estrecho de Gibraltar.* 3ª Edición 2013.
- Mcdonnell Douglas Astronautics Company. *United States Air Force Stability and Control DATCOM*. 1977.
- Ángel A. Rodríguez Sevillano. *Aerodinámica,2ºAV, Plan 2002*, Curso 2003-2004
- Great Owl Publishing. *Surfaces Vortex Lattice Module User Guide.* 2009.
- *APC Propeller Performance Data*

Anexo 1: Código de comunicación XFOIL-Python

```
import subprocess as sp
import os
import shutil
import sys
import string

def Xfoil(NACA,Re,Mach,aseqi=-5,aseqf=5,aseqstep=1): #Declaramos como función
    NACA=str(NACA) #convierte a string las entradas
    Re=str(Re)
    Mach=str(Mach)
    aseqi=str(aseqi)
    aseqf=str(aseqf)
    aseqstep=str(aseqstep)
    filename=NACA+'.txt' #Nombre de la función de guardado

    try:                #Elimina registros antiguos
        os.remove(filename)
    except OSError:
        pass
    filename='dump'+NACA
    try:
        os.remove(filename)
    except OSError:
        pass

    with sp.Popen(['xfoil.exe'],stdin=sp.PIPE,stdout=sp.DEVNULL,
       bufsize=1, universal_newlines=True) as p:
    #Esto corre Xfoil si está en la misma carpeta
        p.stdin.write('naca'+NACA+'\n') #Comandos para introducir a Xfoil
        p.stdin.write('oper\n')
        p.stdin.write('re'+Re+'\n')
        p.stdin.write('mach'+Mach+'\n')
        p.stdin.write('visc\n')
        p.stdin.write('pacc\n')
        p.stdin.write(NACA+'.txt\n')
        p.stdin.write('dump'+NACA+'\n')
        p.stdin.write('aseq'+aseqi+' '+aseqf+' '+aseqstep+'\n')
        p.stdin.write('\n')
        p.stdin.write('quit\n')
```

Anexo 2: Código de interpretación de resultados de XFOIL

```
#importación de modulos
import numpy as np
import matplotlib.pyplot as plt
from sympy import *
init_printing(use_latex=True)
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.tri
from matplotlib import cm
import string
from scipy import interpolate
from scipy.integrate import quad

def importarXfoildata(NACA): #Función importación
    """Importa los datos del perfil resultado de xfoil, deben estar dentro de
    una carpeta llamada Perfiles en la misma carpeta"""
    NACA=str(NACA)
    data=np.loadtxt('Perfiles/'+NACA+'.txt',skiprows=12) #importar todos los
    datos y guardarlos por variable
    alphavector=data[:,0] #importa cada columna en su variable.
    alphavectorradianes=alphavector*(np.pi/180)
    Clvector=data[:,1]
    Cdvector=data[:,2]
    Cdpvector=data[:,3]
    Cmvector=data[:,4]

    #Interpolar Cl, Cd para convertirlos en función
    Clfuncion=interpolate.InterpolatedUnivariateSpline(alphavectorradianes,
    Clvector,k=1)
    Cdfuncion=interpolate.InterpolatedUnivariateSpline(alphavectorradianes,
    Cdvector,k=1)
    CdvsCl=interpolate.InterpolatedUnivariateSpline(Clvector,Cdvector,k=1)
    #Interpola alpha sobre Cl para obtener alpha0 en radianes
    alpha0funcion=interpolate.InterpolatedUnivariateSpline(Clvector,
    alphavectorradianes,k=1)
    alpha0=float(alpha0funcion(0))
    #función para obtener la pendiente Clalpha, importante que n y -n estén en
    la zona linear

    n=4
    n=n*(np.pi/180)
    Clalpha=(Clfuncion(n)-Clfuncion(-n))/((n-(-n))) #Obtenemos Clalpha en 1/rad

    return Clfuncion,Cdfuncion,CdvsCl,alpha0,Clalpha #el resultado es Cl, Cd
    como función de Alpha, alpha0, Clalpha y Cd función de Cl
```

Anexo 3: Código VLM completo.

```
#importación de modulos
import numpy as np
import matplotlib.pyplot as plt
from sympy import *
init_printing(use_latex=True)
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.tri
from matplotlib import cm
from scipy.spatial import ConvexHull
import subprocess as sp
import os
import shutil
import sys
import string
from scipy import interpolate
from scipy.integrate import quad
```

```
#definición de funciones
def importarXfoildata(NACA):
    """Importa los datos del perfil resultado de xfoil, deben estar dentro de
una carpeta llamada Perfiles en la misma carpeta"""
    NACA=str(NACA)
    data=np.loadtxt('Perfiles/'+NACA+'.txt',skiprows=12) #Importa todos los dat
os y guardarlos por variable
    alphavector=data[:,0]
    alphavectorradianes=alphavector*(np.pi/180)
    Clvector=data[:,1]
    Cdvector=data[:,2]
    Cdpvector=data[:,3]
    Cmvector=data[:,4]

    #interpolar cl, cd para convertirlos en función
    Clfuncion=interpolate.InterpolatedUnivariateSpline(alphavectorradianes,
Clvector,k=1)
    Cdfuncion=interpolate.InterpolatedUnivariateSpline(alphavectorradianes,
Cdvector,k=1)
    CdvsCl=interpolate.InterpolatedUnivariateSpline(Clvector,Cdvector,k=1)
    #interpola alpha sobre Cl para obtener alpha0 en radianes
    alpha0funcion=interpolate.InterpolatedUnivariateSpline(Clvector,
alphavectorradianes,k=1)
    alpha0=float(alpha0funcion(0))

    #función para obtener la pendiente Clalpha, importante que n y -n estén en
la zona linear
    n=4
    n=n*(np.pi/180)
    Clalpha=(Clfuncion(n)-Clfuncion(-n))/((n-(-n))) #da Clalpha en 1/rad
```

```
    return Clfuncion,Cdfuncion,CdvsCl,alpha0,Clalpha
    #el resultado es Cl,Cd como función y alpha0 y Clalpha

def aguilacalzada(angulocentral,angulolateral,b):
    """Calcula la geometría del ala basada en el águila calzada,
    las entradas son los angulos de movimiento de las
    articulaciones de las alas y la envergadura del águila.
    """
    y=symbols('y')
    x=symbols('x')

    x=-7*y**6 #geometría de la parte delantera sin flecha
    z=x.subs(y,b/2)-0.015*(1+cos(2*(np.pi*y)))
    #geometría de la parte trasera sin flecha

    ccuartos=x-(x-z)/4 #línea de c cuartos
    c=(x-z) #cuerda
    articulacion=0.25

    #se crean unos huesos en la línea c/4, centrales y laterales
    huesocentral=-tan(rad(angulocentral))*abs(y)+ccuartos
    huesolateral=-tan(rad(angulolateral))*abs(y)+ccuartos-(tan(rad(angulocentra
l))-tan(rad(angulolateral)))*abs(articulacion)

    #convertir función simbólica con función numérica.
    x=lambdify(y,x)
    z=lambdify(y,z)
    ccuartos=lambdify(y,ccuartos)
    cplot=lambdify(y,c)
    huesocentral=lambdify(y,huesocentral)
    huesolateral=lambdify(y,huesolateral)

    def hueso(y):
        '''Convierte las funciones de huesocentral y huesolateral
        en una sola'''
        x=np.zeros(y.size)
        for i in range(y.size):
            if abs(y[i])<=articulacion :
                x[i]=huesocentral(y[i])
            else:
                x[i]=huesolateral(y[i])
        return x
    y=np.linspace(-b/2,b/2,100)

    #borde de ataque y borde de salida sumando cuerda a los huesos.
    bordeataque=hueso(y)+cplot(y)/4
    bordesalida=hueso(y)-3*cplot(y)/4

    bordeataque=interpolate.InterpolatedUnivariateSpline(y,bordeataque,k=1)
```

```
    bordesalida=interpolate.InterpolatedUnivariateSpline(y,bordesalida,k=1)
    return c,cplot,bordeataque,bordesalida
```

```
grafica=plt.figure()
for jk in [-10,0,10]: #Bucle para girar articulaciones.
    for ik in [-10,0,10]:
        #-------------
        V=10
        rho=1.225
        Q=0.5*(V**2)*rho
        #--------- Características del perfil
        NACA=4412
        Clfuncion,Cdfuncion,CdvsCl,alpha0,Clalpha=importarXfoildata(NACA)
        Clα=Clalpha #en 1/radianes
        Cl0=float(Clfuncion(0)) #da igual en grados o en radianes
        α0=alpha0 # cambiado a radianes
        Cdplot=Cdfuncion #como función de radianes
        #------------ Definición de la geometría
        b=1 #envergadura en metros
        angulocentral=jk
        angulolateral=ik
        y=symbols("y")

        c,cplot,bordeataque,bordesalida=aguilacalzada(angulocentral,
        angulolateral,b)
        twistroot=0 #torsión en la raiz
        twisttip=ik #tornsión en la punta
        #la torsión se trata de forma lineal.

        #-------------- Parámetro de mallado
        paneles=8 # nº paneles por semiala
        #---------------
        #------- Ángulos de ataque para estudio
        alphai=-4 #Alpha inicial, en grados
        alphaf=8  #Alpha final, en grados
        #-------
        #---------Bloque para calcular el área
        caux=c
        cplot=lambdify(y,caux)
        S,errordeS=quad(cplot,-b/2,b/2)
        print('Superficie alar=',round(S,4),'m^2')
        #---------
        #---------Bloque para calculo de la cuerda media del ala
        def cplotcuadrado(x):
            return cplot(x)*cplot(x)
        integralccuadrado,errordec=quad(cplotcuadrado,-b/2,b/2)
        cmedia=(1/S)*integralccuadrado
        #---------
```

```
#---------Bloque para calcular alargamiento
AR=b**2/S
print('AR=',round(AR,4))
#---------
#---------Bloque para el cálculo de la línea c/4 y c3/4
def Ccuartosplot(yn):
    x=np.zeros(yn.size)
    for i in range(yn.size):
        x[i]=bordeataque(yn[i])-cplot(yn[i])/4
    return x

def Ctrescuartosplot(yn):
    x=np.zeros(yn.size)
    for i in range(yn.size):
        x[i]=bordeataque(yn[i])-3*cplot(yn[i])/4
    return x
#---------
#---------Bloque para definir torsión
twist=(twisttip-twistroot)*abs(y)/(b/2)+twistroot # torsión lineal
plottwist=lambdify(y,twist)
#---------
#----------Representación gráfica de la vista en planta
grafica.add_subplot(6,1,1)
x=np.linspace(-b/2,b/2,100)
plotbordeataque=bordeataque
plotbordesalida=bordesalida
plt.plot(x,plotbordeataque(x))
plt.plot(x,plotbordesalida(x))
plt.plot(x,Ccuartosplot(x),'--')
plt.plot(x, Ctrescuartosplot(x),'--')
plt.axvline(0,c='black', ls= 'dashed')
plt.axis('equal')
plt.title('Vista en planta')
plt.ylabel('x metros')
plt.xlabel('y metros')
#-----------------
#-------Obtención de puntos de control y matriz C de VLM
Δy=(b/(2*paneles))#longitud en envergadura de cada panel

def Cmatrixfunc(paneles=paneles,b=b):
    '''Calcula la matriz C de VLM, el vector de torsión,
    las cuerdas medias de los paneles, vector torsión.
    '''
    #Parte derecha
    yd=np.linspace(0,b/2,paneles+1)
    for ii in range(paneles+1):
        plt.axvline(yd[ii],c='pink', ls= 'dashed')
        plt.axvline(-yd[ii],c='pink', ls= 'dashed')
```

```
#coordenadas de los puntos de control de cada panel
ym=yd[0:paneles]+(b/2)*(1/paneles)*(1/2)
xm=-(Ctrescuartosplot(ym))

y1n=yd[0:paneles] #puntos en la línea c/4
y2n=yd[1:paneles+1]
x1n=-(Ccuartosplot(y1n))
x2n=-(Ccuartosplot(y2n))

plt.plot(y1n,-x1n,'o',color='black')
plt.plot(y2n,-x2n,'o',color='black')
plt.plot(ym,-xm,'o',color='brown')

#----torsión
twistvector=np.zeros(paneles)
for iii in range(paneles):
    twistvector[iii]=plottwist(ym[iii])*np.pi/180
#devuelve vector de torsión en radianes

#Cuerda en los puntos de control
cavvector=np.zeros(paneles)
xccuartosvector=np.zeros(paneles)
for jjj in range(paneles):
    cavvector[jjj]=cplot(ym[jjj])
    #para el cálculo de CM
    xccuartosvector[jjj]=-(bordeataque(ym[jjj])-cplot(ym[jjj])/4)


#------------------calculo matriz C
#crear matrices para rellenarlas posteriormente
amatrix=np.zeros((paneles,paneles))
bmatrix=np.zeros((paneles,paneles))
cmatrix=np.zeros((paneles,paneles))
dmatrix=np.zeros((paneles,paneles))
gmatrix=np.zeros((paneles,paneles))
hmatrix=np.zeros((paneles,paneles))
ematrix=np.zeros((paneles,paneles))
fmatrix=np.zeros((paneles,paneles))
Cmatrixderecha=np.zeros((paneles,paneles))

for kk in range(paneles):
    for ll in range(paneles):
  #definir punto por punto las distintas variables
        amatrix[kk,ll]=xm[kk]-x1n[ll]
        bmatrix[kk,ll]=ym[kk]-y1n[ll]
        cmatrix[kk,ll]=xm[kk]-x2n[ll]
        dmatrix[kk,ll]=ym[kk]-y2n[ll]
        gmatrix[kk,ll]=x2n[kk]-x1n[kk]
        hmatrix[kk,ll]=y2n[kk]-y1n[kk]
        ematrix[kk,ll]=sqrt(amatrix[kk,ll]**2+bmatrix[kk,ll]**2)
```

```
                    fmatrix[kk,ll]=sqrt(cmatrix[kk,ll]**2+dmatrix[kk,ll]**2)
                    a,b,c,d,e,f,g,h=amatrix[kk,ll],bmatrix[kk,ll],cmatrix[kk,ll
],dmatrix[kk,ll],ematrix[kk,ll],fmatrix[kk,ll],gmatrix[kk,ll],hmatrix[kk,ll]

                    Cmatrixderecha[kk,ll]=((1/(a*d-c*b))*(((g*a+h*b)/e)-((g*c+h
*d)/f))-(1/b)*(1+a/e)+(1/d)*(1+c/f))*(1/(2*Clα))
            #----------------
            #Parte izquierda
            #coordenadas de los puntos de control de cada panel
            ymi=ym
            xmi=xm

            y1ni=-y2n
            y2ni=-y1n

            x1ni=x2n
            x2ni=x1n

            plt.plot(y1ni,-x1ni,'o',color='black')
            plt.plot(y2ni,-x2ni,'o',color='black')
            plt.plot(ymi,-xmi,'o',color='brown')

            #------------------calculo matriz C
            #crear matrices para rellenarlas posteriormente
            amatrixi=np.zeros((paneles,paneles))
            bmatrixi=np.zeros((paneles,paneles))
            cmatrixi=np.zeros((paneles,paneles))
            dmatrixi=np.zeros((paneles,paneles))
            gmatrixi=np.zeros((paneles,paneles))
            hmatrixi=np.zeros((paneles,paneles))
            ematrixi=np.zeros((paneles,paneles))
            fmatrixi=np.zeros((paneles,paneles))
            Cmatrixi=np.zeros((paneles,paneles))

            for kk in range(paneles):
                for ll in range(paneles):
                    amatrixi[kk,ll]=xmi[kk]-x1ni[ll]
                    bmatrixi[kk,ll]=ymi[kk]-y1ni[ll]
                    cmatrixi[kk,ll]=xmi[kk]-x2ni[ll]
                    dmatrixi[kk,ll]=ymi[kk]-y2ni[ll]
                    gmatrixi[kk,ll]=x2ni[kk]-x1ni[kk]
                    hmatrixi[kk,ll]=y2ni[kk]-y1ni[kk]
                    ematrixi[kk,ll]=sqrt(amatrixi[kk,ll]**2+bmatrixi[kk,ll]**2)
                    fmatrixi[kk,ll]=sqrt(cmatrixi[kk,ll]**2+dmatrixi[kk,ll]**2)
                    a,b,c,d,e,f,g,h=amatrixi[kk,ll],bmatrixi[kk,ll],cmatrixi[kk
,ll],dmatrixi[kk,ll],ematrixi[kk,ll],fmatrixi[kk,ll],gmatrixi[kk,ll],hmatrixi[k
k,ll]

                    Cmatrixi[kk,ll]=((1/(a*d-c*b))*(((g*a+h*b)/e)-((g*c+h*d)/f)
)-(1/b)*(1+a/e)+(1/d)*(1+c/f))*(1/(2*Clα))
```

```
    Cmatrix=Cmatrixderecha+Cmatrixi
    return Cmatrix,twistvector,cavvector,ym

Cmatrix,twistvector,cavvector,ym=Cmatrixfunc()
#--------------


#-------------Definición de vector w y Cálculo de CL
alphavectorgrados=np.linspace(alphai,alphaf,51)
alphavector=alphavectorgrados*np.pi/180 #alpha en radianes

#creamos las matrices para luego rellenarlas
Clpaneles=np.zeros((paneles,alphavector.size))
Cdpaneles=np.zeros((paneles,alphavector.size))
Cmpaneles=np.zeros((paneles,alphavector.size))
CL=np.zeros(alphavector.size)
CD=np.zeros(alphavector.size)
CM=np.zeros(alphavector.size)
L=np.zeros(alphavector.size)

for hh in range(alphavector.size):
    wvector=np.zeros(paneles)
    for k in range(paneles):
        #vector condiciones de contorno.
        wvector[k]=-V*(alphavector[hh]+twistvector[k])
    Γn=np.linalg.solve(Cmatrix,wvector)
    Clpaneles[:,hh]=2*Γn/(V*cavvector)
    Clpaneles[:,hh]=Clpaneles[:,hh]-α0*Clα #se le añade el efecto de α0
    Cmpaneles[:,hh]=Clpaneles[:,hh]*xccuartosvector/cmedia
    Cdpaneles[:,hh]=CdvsCl(Clpaneles[:,hh])
      #Se calcula Cd usando las gráficas de XFOIL

    CL[hh]=(2/S)*sum(Δy*cavvector*Clpaneles[:,hh])
      #coeficiente de sustentación del ala.
    CD[hh]=(2/S)*sum(Δy*cavvector*Cdpaneles[:,hh])
      #coeficiente de resistencia
    CM[hh]=(2/S)*sum(Δy*cavvector*Cmpaneles[:,hh])
      #coeficiente de momento de cabeceo

    L[hh]=Q*CL[hh]*S
CLalpha=(CL[-1]-CL[0])/(alphavector[-1]-alphavector[0])
CLalphaporgrado=CLalpha*np.pi/180
CLvCD=CL/CD
#------------
#---------Cálculo de CL0
CLfuncion=interpolate.InterpolatedUnivariateSpline(alphavector,CL,k=1)
CL0=float(CLfuncion(0))
#-----
```

```
#-----------Representación gráfica de la polar
grafica.add_subplot(6,1,2)
plt.plot(alphavectorgrados,CL,
label=str(angulocentral)+str(angulolateral))

grafica.add_subplot(6,1,3)
plt.plot(alphavectorgrados,CD,
label=str(angulocentral)+str(angulolateral))

grafica.add_subplot(6,1,4)
plt.plot(alphavectorgrados,CLvCD,
label=str(angulocentral)+str(angulolateral))

grafica.add_subplot(6,1,5)
plt.plot(alphavectorgrados,CM,
label=str(angulocentral)+str(angulolateral))

grafica.add_subplot(6,1,6)
plt.plot(CD,CL,
label=str(angulocentral)+str(angulolateral))


#-----------

#-----------Bloque para guardar los datos
filename=str(NACA)+'_'+str(angulocentral)+'_'+str(angulolateral)
try:
    os.remove(filename)
except OSError:
    pass

np.save('Alas/CL'+'_'+filename,CL)
np.save('Alas/CD'+'_'+filename,CD)
np.save('Alas/CLvCD'+'_'+filename,CLvCD)

Resultados=np.array([    #matriz resumen de resultados
    ['CLα por radián',round(CLalpha,4)],
    ['CLα por grado',round(CLalphaporgrado,4)],
    ['CL0',round(CL0,4)]])

np.save('Alas/Resultados'+'_'+filename,Resultados)

#-----------Print de variables
print('CLα=',round(CLalpha,4),'por radián')
print('CLα=',round(CLalphaporgrado,4), 'por grado')
print('CL0=',round(CL0,4))
#print('α0=',round(α0ala,4),'°')
print('Perfil  ','Clα=',round(Clα,4),'por radián')
#------
```

```
#-----Parámetros para la representación de las gráficas.
grafica.add_subplot(6,1,2)
plt.grid()
plt.title('CL v α ')
plt.ylabel('CL')
plt.xlabel('α grados')
plt.legend(fontsize='x-large')

grafica.add_subplot(6,1,3)
plt.grid()
plt.title('CD v α ')
plt.ylabel('CD')
plt.xlabel('α grados')
plt.legend(fontsize='x-large')

grafica.add_subplot(6,1,4)
plt.grid()
plt.title('CL/CD v α ')
plt.ylabel('CL/CD')
plt.xlabel('α grados')
plt.legend(fontsize='x-large')

grafica.add_subplot(6,1,5)
plt.grid()
plt.title('CM v α '+str(NACA))
plt.ylabel('CM')
plt.xlabel('α grados')
plt.legend(fontsize='x-large')

grafica.add_subplot(6,1,6)
plt.grid()
plt.title('CL v CD ')
plt.ylabel('CL')
plt.xlabel('CD')
plt.legend(fontsize='x-large')


grafica.set_size_inches(8, 64) #Define el tamaño de imagen.

plt.savefig('Alas/comparacion'+filename+'.png') #Guarda la gráfica como imagen.
```

```
#Actuaciones de planeo.
grafica4=plt.figure() #Crea nueva figura.
W=9.81*m #Peso
γD=np.arctan(1/CLvCD) #ángulo de planeo en radianes
γDgrados=γD*(180/np.pi) #angulo de planeo en grados
#Velocidad de planeo
Vplaneo=np.sqrt(np.sqrt(((2*np.cos(γD)*W)/(rho*CL*S))**2))
#Velocidad de descenso
VD=(CD/CL**1.5)*np.sqrt(2*W/(rho*S))*(np.cos(γD)**1.5)

#Representación gráfica.
grafica4.add_subplot(3,1,1)
plt.plot(alphavectorgrados[20:],γDgrados[20:])
plt.grid()
plt.title('γD v α ')
plt.ylabel('γD grados')
plt.xlabel('α grados')

grafica4.add_subplot(3,1,2)
plt.plot(alphavectorgrados[20:],Vplaneo[20:])
plt.grid()
plt.title('V v α ')
plt.ylabel('V')
plt.xlabel('α grados')

grafica4.add_subplot(3,1,3)
plt.plot(alphavectorgrados[20:],VD[20:])
plt.grid()
plt.title('VD v α ')
plt.ylabel('VD')
plt.xlabel('α grados')

grafica4.set_size_inches(5, 35)
```

```
#Módulo para actuaciones de crucero.

grafica2=plt.figure() #Se crea nueva gráfica

CLcrucero=CL[7:] #Escogemos los puntos lejos de la zona de singularidad.
Vcrucero=np.sqrt((W/(0.5*rho*S))/CLcrucero) #Velocidad crucero
Pr=0.5*rho*(V**3)*S*CD[7:] #Potencia requerida

grafica2.add_subplot(1,2,1)
plt.plot(Vcrucero,Pr)
plt.grid()
plt.title('Pr v V ')
plt.ylabel('Pr (W)')
plt.xlabel('V (m/2)')

grafica2.add_subplot(1,2,2)
plt.plot(alphavectorgrados[7:],Pr)
plt.title('Pr v α ')
plt.ylabel('Pr (W)')
plt.xlabel('α grados')

grafica2.set_size_inches(15, 4)


#Módulo para el vuelo estacionario de subida.
#Velocidades y potencias para la hélice 10x10E y 3000RPM
#(introducida manualmente)

V3000=np.array([ 0.,  0.536448,  1.1176  ,  1.654048,  2.190496,  2.726944,
        3.308096,  3.844544,  4.380992,  4.91744 ,  5.498592,  6.03504 ,
        6.571488,  7.107936,  7.689088,  8.225536,  8.761984,  9.343136,
        9.879584, 10.416032, 10.95248 , 11.533632, 12.07008 , 12.606528,
       13.142976, 13.724128, 14.260576, 14.797024, 15.378176, 15.914624])
PE3000=np.array([0.e+00, 7.210e-02, 1.410e-01, 2.065e-01, 2.688e-01, 3.276e-01,
       3.831e-01, 4.353e-01, 4.842e-01, 5.299e-01, 5.725e-01, 6.120e-01,
       6.485e-01, 6.822e-01, 7.131e-01, 7.406e-01, 7.647e-01, 7.855e-01,
       8.033e-01, 8.177e-01, 8.292e-01, 8.381e-01, 8.452e-01, 8.509e-01,
       8.544e-01, 8.552e-01, 8.478e-01, 8.210e-01, 7.313e-01, 8.000e-04])

PD=PE3000*(20) #Potencia disponible

PDfuncion=interpolate.InterpolatedUnivariateSpline(V3000,PD,k=1)
Prfuncion=interpolate.InterpolatedUnivariateSpline(
list(reversed(Vcrucero)),list(reversed(Pr)),k=1)

Vrango=np.linspace(9.8,15.5,50)
```

```
grafica=plt.figure()

plt.plot(V3000,PD,label='Pot. disp.')
plt.plot(Vcrucero,Pr,label='Pot. nec.')
plt.grid()
plt.title('Potencia frente a velocidad')
plt.ylabel('Potencia (W)')
plt.xlim(9,16)
plt.xlabel('V(m/s)')
plt.legend()

grafica2=plt.figure()
plt.plot(Vrango,(PDfuncion(Vrango)-Prfuncion(Vrango))/W) #Velocidad subida

plt.grid()
plt.title('Velocidad subida')
plt.ylabel('Vs(m/s)')
plt.xlabel('V(m/s)')
```

```
#Estabilidad estática longitudinal.
xcg=0.0568 #centro de gravedad
N0=(xcg/cmedia-CM/CL) #punto neutro con mando fijo
H0=N0-xcg/cmedia #margen estático con mando fijo

grafica5=plt.figure()
grafica5.add_subplot(1,2,1)
plt.plot(alphavectorgrados,N0)
plt.grid()
plt.title('Punto neutro con mando fijo ')
plt.ylabel('N0')
plt.xlabel('α grados')

grafica5.add_subplot(1,2,2)
plt.plot(alphavectorgrados,H0)
plt.grid()
plt.title('Margen estático con mandos fijos')
plt.ylabel('H0')
plt.xlabel('α grados')

grafica5.set_size_inches(11, 4)
```

```
#Envolvente de vuelo
Clmax=1.4452 #del perfil

CLmax=float(0.9*Clmax*cos((atan((xccuartosvector[-1]-xccuartosvector[0])/0.5)))
) #del ala

hstall=np.array([0,75,150,225,300,375,450]) #distribución de alturas
#distribución de densidades
rhostall=np.array([1.225,1.2162,1.20746,1.19876,1.19011,1.18150,1.17295])
Vstall=np.sqrt(((2*W)/(rhostall*S*CLmax))) #velocidad de pérdida
Vmax=np.array([15.868,15.868,15.868,15.868,15.868,15.868,15.868])
#velocidad máxima (introducida manualmente)

plt.plot(Vstall,hstall,label='Vstall')
plt.plot(Vmax,hstall,label='Vmax')
plt.axhline(450,c='black', ls= 'dashed', label='techo de vuelo')
plt.title('Envolvente de vuelo')
plt.ylabel('Altura (m)')
plt.xlabel('V (m/s)')
plt.legend(fontsize='large')
```

Anexo 4: Código de representación 3-D

```
#importación de modulos
import numpy as np
import matplotlib.pyplot as plt
from sympy import *
init_printing(use_latex=True)
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.tri
from matplotlib import cm
from scipy.spatial import ConvexHull
from scipy import interpolate
from scipy.integrate import quad

NACA='2412' #Escribir como string el número del perfil NACA de 4 dígitos (solo
acepta NACA de 4 números)

y=symbols("y")
b=1
c,cplot,bordeataque,bordesalida=aguilacalzada(-0,0,b) #Definido en el Anexo 5.
plotbordeataque=bordeataque
plotbordesalida=bordesalida

x=np.linspace(-b/2,b/2,100) #se van a utilizar para la representación 100 punto
s en el eje x

def Nacafunc(NACA,c):
#función para crear la geometría de perfiles NACA de 4 dígitos

    M=float(NACA[0])
    P=float(NACA[1])
    XX=float(NACA[2:4])

    xp=symbols('xp') #Variable simbólica, x del perfil.
    m=M/100
    p=P/10
    t=XX/100
    #línea media
    #el perfil NACA de 4 dígitos cambia las expresiones polinómicas en el punto
p, posición de la ordenada máxima de la cuerda media.
    ycfront=((m/p**2)*(2*p*(xp/c)-(xp/c)**2))*c
    ycback=((m/(1-p)**2)*(1-2*p+2*p*(xp/c)-(xp/c)**2))*c
    dycdxfront=(2*m/p**2)*(p-(xp/c)) #derivadas de la cuerda media.
    dycdxback=(2*m/(1-p)**2)*(p-(xp/c))
    #distribución de espesores
    a0=0.2969
    a1=-0.126
    a2=-0.3516
    a3=0.2843
    a4=-0.1015
    yt=(5*t)*(a0*(xp/c)**0.5+a1*(xp/c)+a2*(xp/c)**2+a3*(xp/c)**3+a4*(xp/c)**4)*
c
    #la superficie se divide en 4 zonas, delantera, trasera, extradós, intradós
, cada una se define por 100 puntos
```

```
    #upper surface
    xufront=(xp-yt*sin(atan(dycdxfront)))
    yufront=ycfront+yt*cos(atan(dycdxfront))
    xuback=(xp-yt*sin(atan(dycdxback)))
    yuback=ycback+yt*cos(atan(dycdxback))
    #lower surface
    xlfront=(xp+yt*sin(atan(dycdxfront)))
    ylfront=ycfront-yt*cos(atan(dycdxfront))
    xlback=(xp+yt*sin(atan(dycdxback)))
    ylback=ycback-yt*cos(atan(dycdxback))

    xbackup=np.linspace(c,p*c,100) #Definición de la distribución de x
    xfrontup=np.linspace(p*c,0,100)
    xfrontdown=np.linspace(0,p*c,100)
    xbackdown=np.linspace(p*c,c,100)

    plotycfront=lambdify(xp,ycfront) #Pasar funciones de simbólicas a numéricas
    plotycback=lambdify(xp,ycback)
    plotxufront=lambdify(xp,xufront)
    plotyufront=lambdify(xp,yufront)
    plotxuback=lambdify(xp,xuback)
    plotyuback=lambdify(xp,yuback)
    plotxlfront=lambdify(xp,xlfront)
    plotylfront=lambdify(xp,ylfront)
    plotxlback=lambdify(xp,xlback)
    plotylback=lambdify(xp,ylback)

    x=np.array(plotxuback(xbackup)) # en ejes avión es y

    x=np.concatenate((x,plotxufront(xfrontup),plotxlfront(xfrontdown),plotxlbac
k(xbackdown))) # Se concatenan las cuatro zonas distintas de x.

    y=np.array(plotyuback(xbackup)) # en ejes avión es z

    y=np.concatenate((y,plotyufront(xfrontup),plotylfront(xfrontdown),plotylbac
k(xbackdown))) # Se concatenan las cuatro zonas distintas de y.

    x=-x # para adecuarse a la dirección de los ejes del avión
    return x,y #la función devuelve las coordenadas de los puntos ordenados de
extradós x=c a intradós x=c antihorario.

#Generar los puntos de la superficie de la aeronave
xpoints=np.linspace(-b/2+0.001,b/2-0.001,100) # puntos a lo largo de x para la
representación, se evitan las zonas con c=0 para evitar singularidades.
yba=plotbordeataque(xpoints) # particularización en los puntos del eje x
ybs=plotbordesalida(xpoints) # particularización en los puntos del eje x
#la geometría del borde de ataque y borde de salida define la cuerda del perfil
c=-(plotbordesalida(xpoints)-plotbordeataque(xpoints))
```

```
#Este bucle define los arrays x,y,z con las coordenadas de todos los puntos de
la aeronave
for ii in range(xpoints.size):
    yaux,zaux=Nacafunc(NACA,c[ii]) #Se llama a la función NACAfunc para definir
los puntos del perfil para esa posición de x
    yaux=yaux+yba[ii]
    if ii==0:
        x=np.array([xpoints[ii]])
        y=yaux
        z=zaux
        for jj in range(4*xpoints.size-1):
            xaux=np.array([xpoints[ii]])
            x=np.concatenate((x,xaux))
    else:

        y=np.concatenate((y,yaux))
        z=np.concatenate((z,zaux))
        for jj in range(4*xpoints.size):
            xaux=np.array([xpoints[ii]])
            x=np.concatenate((x,xaux))

# Representación esquemática 3D de la aeronave, utilizando puntos

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter3D(x, y, z)
ax.set_xlim(-0.75,0.75)
ax.set_ylim(-0.75,0.75)
ax.set_zlim(-0.75,0.75)
fig.set_size_inches(20,20)
ax.invert_yaxis()
```

Anexo 5: Código de definición de la geometría alar.

```
import numpy as np #importación de modulos
import matplotlib.pyplot as plt
from sympy import *
init_printing(use_latex=True)
from matplotlib import cm
from scipy import interpolate

def aguilacalzada(angulocentral,angulolateral,b):
    """Calcula la geometría del ala basada en el águila calzada, las entradas
    son los angulos de movimiento de las articulaciones de las alas
    y la envergadura del águila. """
    y=symbols('y') # Declara x e y como variables simbólicas
    x=symbols('x')
    x=-7*y**6 # Geometría del b.a. sin flecha
    z=x.subs(y,b/2)-0.015*(1+cos(2*(np.pi*y))) # Geometría del b.s. sin flecha
    ccuartos=x-(x-z)/4 # Línea de c cuartos
    c=(x-z) # Cuerda
    articulacion=0.25
    # Se crean unos huesos en la línea c/4, centrales y laterales
    huesocentral=-tan(rad(angulocentral))*abs(y)+ccuartos
    huesolateral=-tan(rad(angulolateral))*abs(y)+ccuartos-
(tan(rad(angulocentral))-tan(rad(angulolateral)))*abs(articulacion)
    x=lambdify(y,x) # Convertir función simbólica con función numérica.
    z=lambdify(y,z)
    ccuartos=lambdify(y,ccuartos)
    cplot=lambdify(y,c)
    huesocentral=lambdify(y,huesocentral)
    huesolateral=lambdify(y,huesolateral)
    def hueso(y):
        '''Convierte las funciones de huesocentral y huesolateral
        en una sola'''
        x=np.zeros(y.size)
        for i in range(y.size):
            if abs(y[i])<=articulacion :
                x[i]=huesocentral(y[i])
            else:
                x[i]=huesolateral(y[i])
        return x
    y=np.linspace(-b/2,b/2,100)
    # Borde de ataque y borde de salida sumando la cuerda a los huesos.
    bordeataque=hueso(y)+cplot(y)/4
    bordesalida=hueso(y)-3*cplot(y)/4
    # Convierte los bordes en funciones numéricas
    bordeataque=interpolate.InterpolatedUnivariateSpline(y,bordeataque,k=1)
    bordesalida=interpolate.InterpolatedUnivariateSpline(y,bordesalida,k=1)
    return c, cplot, bordeataque, bordesalida
```