UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Industrial

Desarrollo e implementación de algoritmos avanzados de control mediante PLCs industriales. Aplicación a la automatización y control híbrido de un proceso industrial mediante un autómata PLCnext Technology.

Trabajo Fin de Máster

Máster Universitario en Ingeniería Industrial

AUTOR/A: Gonzalo Fernández, David

Tutor/a: Simarro Fernández, Raúl

CURSO ACADÉMICO: 2022/2023

RESUMEN

En el presente proyecto se ha desarrollado un control predictivo basado en modelo con capacidad de escalabilidad para procesos multivariables. Para su diseño se ha llevado a cabo la identificación y modelado de un sistema multivariable, seguido de su validación mediante simulación. Se ha utilizado el software de cálculo numérico Matlab para la implementación inicial y la optimización de parámetros, y posteriormente se ha transferido el control a un PLC industrial de la marca Phoenix Contact. Para la implementación en el PLC, se ha considerado el uso de la herramienta PLCcoder, pero finalmente se ha optado por desarrollar funciones en texto estructurado en el autómata.

Para demostrar y validar el funcionamiento del sistema propuesto, se ha utilizado una maqueta de helicóptero como sistema multivariable. Además de controlar este sistema, se ha logrado automatizar un sistema compuesto por elementos industriales, como cilindros neumáticos, motores neumáticos y pinzas de agarre, mediante el PLC.

También se ha desarrollado una interfaz hombre-máquina (HMI) que facilita la interacción entre el PLC y los procesos controlados. Esta automatización, combinada con el control de procesos, permite establecer un enfoque híbrido en el PLC industrial.

Se ha eligido un PLC de la marca Phoenix Contact debido a su ecosistema de software abierto y su hardware modular, que se adapta a nuestras necesidades.

Palabras clave: control predictivo basado en modelo, sistema multivariable, identificación de sistemas, validación mediante simulación, Matlab, PLC industrial, Phoenix Contact, automatización, control híbrido, HMI.

ABSTRACT

In this project, a scalable model-based predictive control system for multivariable processes has been developed. The design involved the identification and modeling of a multivariable system, followed by its validation through simulation. The numerical computing software Matlab was used for the initial implementation and parameter optimization, and subsequently, control was transferred to an industrial PLC from Phoenix Contact. While the possibility of using the PLCcoder tool for control implementation was considered, ultimately, structured text functions were developed in the controller.

To demonstrate and validate the operation of the proposed system, a helicopter model was used as the multivariable system. In addition to controlling this system, the PLC successfully automated a system consisting of industrial elements such as pneumatic cylinders, pneumatic motors, and grippers.

An HMI (Human-Machine Interface) was also developed to enable interaction between the PLC and the controlled processes. This automation, combined with process control, enables a hybrid approach within the industrial PLC.

A Phoenix Contact PLC was chosen due to its open software ecosystem and modular hardware that can be tailored to our needs.

Keywords: model-based predictive control, multivariable system, system identification, simulation validation, Matlab, industrial PLC, Phoenix Contact, automation, hybrid control, HMI.

RESUM

En el present projecte s'ha desenvolupat un control predictiu basat en model amb capacitat d'escalabilitat per a processos multivariables. Per al seu disseny es va dur a terme la identificació i modelatge d'un sistema multivariable, seguit de la seua validació mitjançant simulació. Es va utilitzar el programari de càlcul numèric Matlab per a la implementació inicial i l'optimització de paràmetres, i posteriorment es va transferir el control a un PLC industrial de la marca Phoenix Contact. Per a la implementació en el PLC, es va considerar l'ús de l'eina PLCcoder, però finalment es va optar per desenvolupar funcions en text estructurat en l'autòmat.

Per a demostrar i validar el funcionament del sistema proposat, es va utilitzar una maqueta d'helicòpter com a sistema multivariable. A més de controlar aquest sistema, es va aconseguir automatitzar un sistema compost per elements industrials, com a cilindres pneumàtics, motors pneumàtics i pinces d'agarre, mitjançant el PLC.

També es va desenvolupar una interfície home-màquina (HMI) que facilita la interacció entre el PLC i els processos controlats. Aquesta automatització, combinada amb el control de processos, permet establir un enfocament híbrid en el PLC industrial.

Es va triar un PLC de la marca Phoenix Contact a causa del seu ecosistema de programari obert i el seu maquinari modular, que s'adapta a les nostres necessitats.

Paraules clau: control predictiu basat en model, sistema multivariable, identificació de sistemes, validació mitjançant simulació, Matlab, PLC industrial, Phoenix Contact, automatització, control híbrid, HMI.

ÍNDICE GENERAL

DOCUMENTO Nº1: MEMORIA

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

DESARROLLO E IMPLEMENTACIÓN DE ALGORITMOS AVANZADOS DE CONTROL MEDIANTE PLCS INDUSTRIALES.
APLICACIÓN A LA AUTOMATIZACIÓN Y CONTROL HÍBRIDO DE UN PROCESO INDUSTRIAL MEDIANTE UN AUTÓMATA PLCNEXT TECHNOLOGY.

DOCUMENTO Nº1: MEMORIA

Capítulo 1

Introducción

En el ámbito de la ingeniería de control, se ha observado un gran avance en los controladores lógicos programables (PLCs), los cuales han evolucionado significativamente en términos de capacidad computacional y capacidad para implementar algoritmos con restricciones temporales estrictas. Esto abre la posibilidad de utilizar estos PLC avanzados para la automatización de sistemas y la implementación de algoritmos de control multivariables de alta complejidad.

En este trabajo académico, se ha utilizado una maqueta de un helicóptero (twin rotor) y el sistema neumático como medio para ilustrar las capacidades de los PLC actuales. Sin embargo, es importante destacar que el objetivo principal no se limita a estos sistemas específicos.

Para lograr este objetivo, se utilizará el PLCNext, un PLC industrial que proporciona una plataforma versátil y confiable para el control de sistemas complejos [10]. El PLCNext se caracteriza por su capacidad de programación en lenguajes de alto nivel y su conectividad con una amplia variedad de dispositivos y sensores.

Asimismo, se utiliza el software de automatización y control CoDeSys [9], ampliamente reconocido en la industria, para el desarrollo de aplicaciones de control en tiempo real. CoDeSys proporciona un entorno de programación intuitivo y potente que facilita la implementación de algoritmos de control avanzados.

La inclusión de elementos industriales neumáticos en la automatización ofrece una solución robusta y confiable para la gestión del PLC industrial. Los sistemas neumáticos son muy utilizados en entornos industriales debido a su capacidad para soportar condiciones adversas, como vibraciones, altas temperaturas y humedad.

En resumen, este trabajo académico destaca la evolución de los PLC hacia máquinas con una alta carga computacional y la capacidad de implementar algoritmos de control con restricciones temporales estrictas. Aunque se ha utilizado el twin rotor y el sistema neumático como demostradores, el enfoque principal de este proyecto es detallar la metodología para implementar esta serie de algoritmos de control multivariables en un PLC industrial.

1.1. Motivación

Este Trabajo Fin de Máster se fundamenta en un desafío propio que consiste en la implementación de un control avanzado en un PLC industrial. El control avanzado seleccionado ha sido un control predictivo basado en modelo, ya que me fascina el hecho de poder prever matemáticamente la acción correcta para llevar a un sistema a un punto deseado utilizando una estrategia de optimización.

La posterior implementación de la automatización de un montaje formado por sistemas industriales neumáticos se añade con el objetivo de demostrar la capacidad actual con la que cuenta un PLC, la cual hace posible el control de ambos sistemas en paralelo.

1.2. Antecedentes académicos

En el transcurso del máster se han estudiado diferentes ramas de la ingeniería industrial, durante el último curso de este, se ha profundizado en el conocimiento de la rama de automatización, robótica y control industrial. En este TFG se intenta demostrar los conocimientos adquiridos durante este año para seguir adentrándonos en el mundo de la automatización y, sobre todo, en el mundo del control complejo de sistemas multivariables.

Gracias a las asignaturas impartidas durante este último año, se ha obtenido la base de conocimientos sobre la que se ha desarrollado este proyecto. En primer lugar, en la asignatura llamada “Control Industrial Avanzado” se ha estudiado el algoritmo de control implementado. Por otro lado, en la asignatura llamada “Automatización Industrial” se han recordado los conocimientos adquiridos durante el grado sobre el funcionamiento de los GRAFCETs [1] (del francés *Graphe Fonctionnel de Commande Etape Transition*) y se nos ha introducido en el funcionamiento de la guía GEMMA (Guía de estudio de los modos de marchas y paradas) para su correcta implementación en nuestros proyectos de automatización. Y para finalizar, aunque todas las asignaturas cursadas han aportado conocimiento empleado en este proyecto, en la asignatura llamada “Implementación de sistemas de control” se ha profundizado sobre la simulación y validación de procesos, la realización de bucles de control y la selección de un periodo adecuado para cada proceso.

Con la idea de expandir estos conocimientos adquiridos y conseguir su implementación sobre un PLC industrial surge la idea de la realización del actual trabajo fin de máster.

1.3. Software y material utilizado

1.3.1. Matlab & Simulink

Matlab, acrónimo de "Matrix Laboratory," es una herramienta de programación y lenguaje de alto nivel ampliamente utilizado en diversas áreas. Su principal ventaja es su facilidad de programación con su propio lenguaje (lenguaje M), es adecuado tanto para principiantes como para expertos. Su sintaxis intuitiva y orientada a matrices simplifica la manipulación de datos y cálculos numéricos complejos, los cuales serán necesarios para la implementación de controles avanzados que se realizarán durante el transcurso de este proyecto. Además, ofrece una amplia gama de funciones y herramientas incorporadas que también se utilizarán para realizar el trabajo de una forma más sencilla, y una comunidad activa que contribuye con recursos adicionales.

Matlab destaca por su utilidad en el ámbito educativo, siendo adoptado por muchas instituciones para enseñar programación, matemáticas y análisis numérico.

Por otro lado, Simulink es una plataforma de simulación y modelado gráfico desarrollada por MathWorks. Su enfoque en el modelado visual mediante bloques interconectados facilita la simulación de sistemas dinámicos complejos en diversas áreas. Ofrece una amplia biblioteca de bloques predefinidos y permite crear bloques personalizados para adaptarse a necesidades específicas. La capacidad de realizar simulaciones en tiempo real es especialmente valiosa en el diseño de sistemas críticos como el sistema estudiado en este proyecto.

Figura 1. Icono software Matlab & Simulink. Fuente: [12]

1.3.2. CoDeSys.

CODESYS es una plataforma de automatización para programar sistemas de control industrial [9]. Se enfoca en el estándar IEC 61131-3, que ofrece cinco lenguajes de programación para PLCs los cuales en este programa se desarrollarán dentro de POUs o "Program Organization Unit". Su interfaz gráfica es intuitiva y cuenta con una amplia biblioteca de funciones predefinidas. Es versátil debido a su compatibilidad con diversos dispositivos como el PLC seleccionado para este proyecto.

El hecho de que CoDeSys permita mediante módulos la implementación del mismo código sobre diferentes dispositivos de diferentes marcas hace que este software se amolde a la perfección sobre uno de los objetivos de este proyecto. Este objetivo es el de desarrollar una base de control que pueda ser migrada a diferentes plataformas hardware (PLCs industriales) sin tener que realizar grandes cambios.

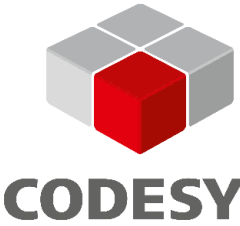

Figura 2. Icono software CoDeSys. Fuente: [13]

1.3.3. PLCnext de Phoenix Contact

El autómata seleccionado es un controlador lógico programable (PLC) que pertenece a la familia PLCnext de Phoenix Contact. PLCnext Technology es el ecosistema para la automatización industrial que consiste en un hardware abierto, un software de ingeniería modular, una comunidad global y el mercado de software digital [10].

Figura 3. Autómata PLCNext

Este PLC cuenta con diversos módulos que son capaces de aportarle diversas funcionalidades.

En primer lugar, cuenta con unos módulos llamados “AXL SE DO16/1” y “AXL SE DI16/1” que aportan a nuestro sistema la capacidad de trabajar hasta con 16 entradas y 16 salidas digitales.

Además, el PLC utilizado, ya que debe controlar un sistema multivariable en el que es necesaria tanto la lectura como escritura de valores analógicos, necesita un módulo llamado “AXL F AI2 AO2 1H” que aporta al sistema la capacidad de trabajar con hasta 2 entradas analógicas y con 2 salidas analógicas. Una característica diferenciadora de este módulo es que sus entradas y salidas analógicas son bipolares entre -10 y +10 voltios, lo que hará nuestra implementación más sencilla al trabajar con un motor que necesita ambos signos en la tensión.

1.3.4. Tarjeta de adquisición de datos

Las tarjetas de adquisición están constituidas por convertidores analógico-digital (A/D) y digital-analógico (D/A) que permiten actuar sobre el proceso y medir su evolución convirtiendo voltajes en números binarios y al contrario.

La tarjeta de adquisición de datos utilizada es la USB-6001, es un dispositivo DAQ de E/S multifunción diseñado por National Instruments (NI), de conexión por USB. Este dispositivo ofrece 8 entradas y 2 salidas analógicas con una resolución de 14 bits y un rango de trabajo de -/+ 10 V.

Las unidades con las que se contaban en el laboratorio se habían integrado en una caja con conectores tipo banana para proporcionar el acceso a las entradas y salidas analógicas.

La utilización de esta tarjeta de adquisición de datos ha sido indispensable para la toma de datos que ha permitido la realización del modelo del proceso y, también ha sido utilizada durante la primera implementación del control utilizando Matlab.

Figura 4. DAQ USB-6001

Capítulo 2

Sistema multivariable

En este capítulo se realizará una explicación paso a paso sobre la metodología utilizada tanto para la identificación como el modelado y control del sistema multivariable, concretamente en este proyecto se ha optado para dicha finalidad por la utilización de una maqueta de helicóptero (Twin rotor).

2.1. Identificación

En primer lugar, se ha decidido estudiar la dinámica del sistema multivariable seleccionado, este estudio nos ha aportado conocimientos sobre su comportamiento y sus características no lineales. Este estudio nos ayudará a la hora de diseñar un controlador predictivo basado en el modelo no lineal en espacio de estados de nuestro "Twin Rotor".

Para la obtención de este modelo es necesario describir matemáticamente el comportamiento de nuestro sistema mediante el uso de ecuaciones. Con este objetivo en mente, se ha decidido dividir el sistema multivariable en dos subsistemas, por un lado, el rotor principal que se encarga del cabeceo y, por otro lado, el rotor de cola que se encarga del giro.

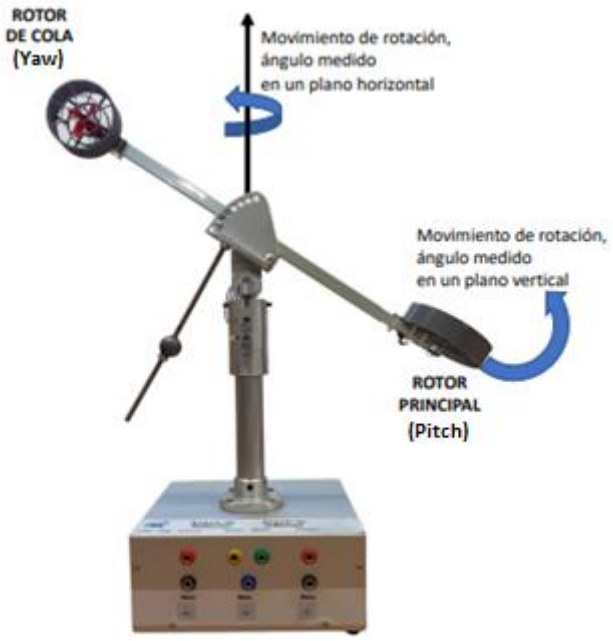

Figura 5. Maqueta "Twin Rotor"

En la **Figura 5** se muestran los movimientos de rotación (Yaw) y cabeceo (Pitch) provocados por el empuje que realizan cada uno de los motores.

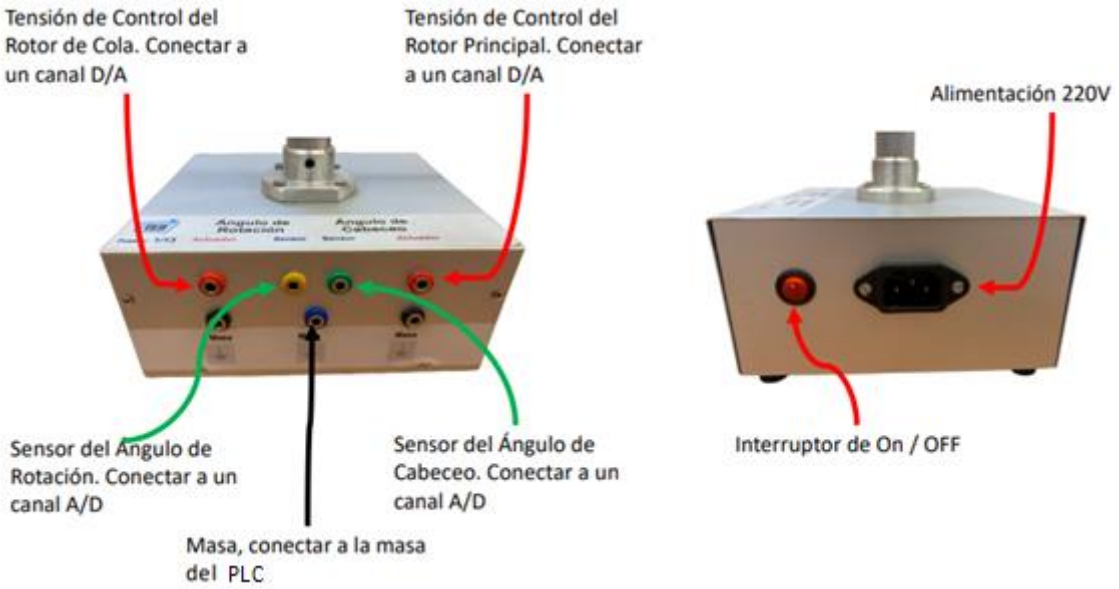

Figura 6. Descripción de E/S del proceso

Para controlar la rotación de ambas hélices, se debe aplicar una tensión sobre los motores correspondientes (variables manipuladas). Los rangos de valores posibles en cada uno de los motores son:

- En el rotor principal con un accionamiento de 0V a 6V. Donde con 0V la hélice no se mueve y con 6V se consigue el régimen de giro máximo.
- El rotor de cola con un accionamiento de −6V a 6V. Donde con 0V la hélice no se mueve, con −6V se consigue el máximo régimen de giro en un sentido y con 6V el máximo régimen de giro en sentido contrario.

Los ángulos de giro (pitch y yaw) se obtienen a través de unos encoders de efecto hall que ofrecen una tensión relacionada con el valor del ángulo correspondiente. Como se puede ver en la **Figura 6**, todas estas señales son accesibles a través de unos conectores y se pueden llevar directamente a las entradas y salidas analógicas de nuestro PLC.

2.2. Modelado

2.2.1. Modelo no lineal

Para el abordaje del modelado de este sistema se ha realizado una serie de simplificaciones. Se ha supuesto un sistema formado por 3 masas (**Figura 7**) ya que los elementos intermedios de este proceso no son vitales. Se considerará positivo el giro que cumpla la regla de la mano derecha en su eje correspondiente.

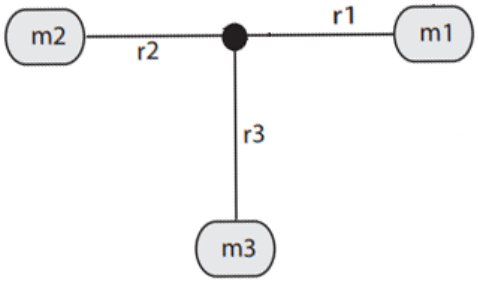

Figura 7. Esquema Modelo 3 masas

Las fuerzas de propulsión ejercidas por cada uno de los motores vienen definidas por las siguientes expresiones obtenidas de forma experimental**:**

- Para el motor de cabeceo (pitch):

$$F_{P_C}(v_c) = P_{c_3} v_c \dot{\alpha} + p_{c_2} v_c^2 + p_{c_1} v_c + p_{c_0} \quad [1]$$

- Para el motor de cola o giro (yaw):

$$F_{P_g}(v_g) = p_{g_1} v_g + p_{c_0} \quad [2]$$

Por otra parte, los coeficientes de fricción de los dos ejes frente a la velocidad angular se pueden describir como:

$$K_{f_c}(\dot{\alpha}) = a_{c_2} \dot{\alpha}^2 + a_{c_0} \quad [3]$$

$$K_{f_g}(\dot{\theta}) = a_{g_0} \quad [4]$$

Los parámetros utilizados en todas las expresiones que definen nuestro sistema se encuentran en la **Tabla 1** presentada a continuación. En ella se encuentran los valores de las masas de nuestro modelo, los radios de giro de cada una de esas masas, los valores de las inercias totales para cada eje, etc.

Tabla 1. Tabla de variables y constantes del modelo

Parámetro	Valor	Unidades	Descripción
m_1	0.0295	Kg	Masa 1 (rotor principal)
m_2	0.0230	Kg	Masa 2 (rotor de cola)
m_3	0.0210	Kg	Masa 3 (contrapeso)
r_1	0.19	m	Radio de giro de la masa 1
r_2	0.17	m	Radio de giro de la masa 2
r_3	0.11	m	Radio de giro de la masa 3
J_c	0.000135	$Kg \cdot m^2$	Inercia total del conjunto de masas en Cabeceo
J_g	0.000108	$Kg \cdot m^2$	Inercia total del conjunto de masas en Guiñada
g	9.81	m/s^2	Constante gravedad
F_{p_c}		N	Fuerza de empuje generada por el rotor principal
p_{c_3}	-0.00025	$N/(V^2 \cdot rad/s)$	Constante 3 del polinomio de ajuste de F_{p_c}
p_{c_2}	-0.00060	N/V^2	Constante 2 del polinomio de ajuste de F_{p_c}
p_{c_1}	0.03260	N/V	Constante 1 del polinomio de ajuste de F_{p_c}
p_{c_0}	0	N	Constante 0 del polinomio de ajuste de F_{p_c}
K_{f_c}		$N \cdot m/(rad/s)$	Coeficiente del par de fricción de Cabeceo
a_{c_2}	-0.00001	$N \cdot m/(rad/s)^3$	Constante 2 del polinomio de ajuste de K_{f_c}
a_{c_0}	0.00020	$N \cdot m/(rad/s)$	Constante 0 del polinomio de ajuste de K_{f_c}
F_{p_g}		N	Fuerza de empuje generada por el rotor de cola
p_{g_1}	0.0017	N/V	Constante 1 del polinomio de ajuste de F_{p_g}
p_{g_0}	0	N	Constante 0 del polinomio de ajuste de F_{p_g}
K_{f_g}		$N \cdot m/(rad/s)$	Coeficiente del par de fricción de Guiñada
a_{g_0}	0.000188	$N \cdot m/(rad/s)$	Constante 0 del polinomio de ajuste de K_{f_g}
v_c		V	Tensión aplicada al rotor principal
v_g		V	Tensión aplicada al rotor de cola
α		rad	Ángulo de cabeceo
Θ		rad	Ángulo de guiñada

Para la obtención de las ecuaciones que definen el comportamiento de nuestro sistema, se deberá abordar el problema desde cada uno de los planos de movimiento realizando un equilibrio de momentos para así poder despejar las variables de interés.

En el plano vertical o de cabeceo (pitch).

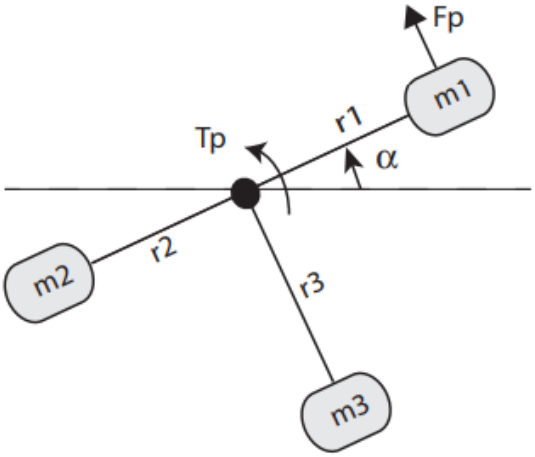

Figura 8. Esquema para modelado del cabeceo

$$J_c \frac{d^2\alpha}{dt} = r_1 \cdot F_{P_c} - r_1 m_1 g \cdot \cos(\alpha) + r_2 m_2 g \cdot \cos(\alpha) - r_3 m_3 g \cdot \sin(\alpha) - K_{f_c} \frac{d\alpha}{dt} \qquad [5]$$

En el plano horizontal o de giro (yaw).

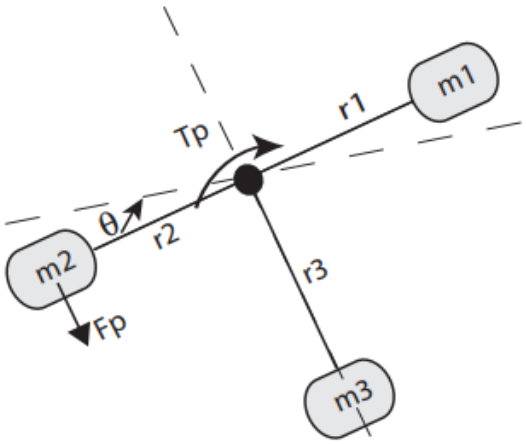

Figura 9. Esquema para modelado de giro

Como se puede observar en este caso la fuerza de la gravedad no afecta al no ser el movimiento en el plano horizontal.

$$J_g \frac{d^2\theta}{dt} = r_2 \cdot F_{P_g} \cdot \cos(\alpha) - K_{f_g} \frac{d\theta}{dt} \quad [6]$$

La implementación de las ecuaciones desarrolladas anteriormente se ha hecho en Matlab con el siguiente código:

```
%% Implementación de las ecuaciones que conforman el sistema.
% En el plano vertical (Pitch)
syms p_alpha p_dalpha p_d2alpha p_pc3 p_pc2 p_pc1 p_pc0 p_ac2 p_ac0 p_r1 p_r2
p_r3 p_m1 p_m2 p_m3 p_g p_Jc p_vc;
eq_pitch_NL=-p_Jc*p_d2alpha   +   p_r1*(p_pc3*p_vc*p_dalpha   +   p_pc2*p_vc^2   +
p_pc1*p_vc+p_pc0)-...
            p_r1*p_m1*p_g*cos(p_alpha)     +     p_r2*p_m2*p_g*cos(p_alpha)     -
p_r3*p_m3*p_g*sin(p_alpha)-...
            (p_ac2*p_dalpha^2+p_ac0)*p_dalpha;
eq_pitch_NL=subs(eq_pitch_NL, [p_pc3 p_pc2 p_pc1 p_pc0 p_ac2 p_ac0 p_r1 p_r2
p_r3 p_m1 p_m2 p_m3 p_g p_Jc], [pc3 pc2 pc1 pc0 ac2 ac0 r1 r2 r3 m1 m2 m3 g
Jc]);


% En el plano horizontal (Yaw)
syms p_theta p_dtheta p_d2theta p_pg1 p_pg0 p_ag0 p_vg p_Jg;
eq_yaw_NL=-p_Jg*p_d2theta      +      p_r2*(p_pg1*p_vg+p_pg0)*cos(p_alpha)      -
p_ag0*p_dtheta;
eq_yaw_NL=subs(eq_yaw_NL, [p_r2 p_pg1 p_pg0 p_ag0 p_Jg], [r2 pg1 pg0 ag0 Jg]);
```

Para obtener los valores teóricos de tensión aplicada necesaria para conseguir mantener nuestro sistema en el equilibrio ($\theta = 0º$ y $\alpha = 0º$), es necesario resolver las ecuaciones de equilibrio de momentos en los diferentes planos.

```
% Obtener Pto Equilibrio (d/dt=0)
alpha_0 = deg2rad(0);
theta_0 = deg2rad(0);
eq_pitch_eq=subs(eq_pitch_NL, [p_dalpha p_d2alpha p_alpha], [0 0 alpha_0]);
eq_yaw_eq=subs(eq_yaw_NL, [p_dtheta p_d2theta p_theta p_alpha], [0 0 theta_0 alpha_0]);


pitch_pto_eq=solve(eq_pitch_eq);
yaw_pto_eq=solve(eq_yaw_eq);


vc_0=double(pitch_pto_eq(1));
vg_0=double(yaw_pto_eq(1));
```

Con este código se consigue el valor de tensiones teóricas a aplicar para mantener nuestro sistema en el equilibrio. Estas tensiones son de 2.83V para el motor de cabeceo (pitch) y de 0V para el motor de giro (yaw).

2.2.2. Modelo lineal

Debido a que la mayoría de las herramientas para el análisis de sistemas y diseño de sistemas de control requieren que el modelo sea lineal, es necesario disponer de métodos para linealizar modelos. Es este proyecto se ha utilizado la serie de Taylor centrada en el punto de equilibrio seleccionado.

$$f(x) = \sum_{k=0}^{\infty} \frac{1}{k!} \frac{d^k f(x)}{dx^k}\Big|_{x_0} (x - x_0)^k \qquad [7]$$

Partiendo de la expresión anterior que ejemplifica el desarrollo de Taylor para una función de una única variable, se deberá implementar para nuestro sistema, que tiene más de una. Ya que todas las derivadas parciales son respecto al incremento de las variables respecto al punto de equilibrio y sabiendo que el punto de equilibrio del proceso a controlar es en el que el "Twin Rotor" se mantiene estático en el aire ($\theta = 0º$ y $\alpha = 0º$), podemos concluir que nuestro valor de θ y α no dependerá de ellas mismas.

- En el plano vertical o de cabeceo (Pitch).

$$r_1 \cdot F_{P_c} - r_1 m_1 g \cdot \cos(\alpha) + r_2 m_2 g \cdot \cos(\alpha) - r_3 m_3 g \cdot \sin(\alpha) - a_{c_2}\dot{\alpha}^2 + a_{c_0}\ \dot{\alpha} - J_c \ddot{\alpha} = 0$$

Donde:

$$F_{P_C}(v_c) = P_{c_3} v_c \dot{\alpha} + p_{c_2} v_c^2 + p_{c_1} v_c + p_{c_0} \quad \text{y} \quad K_{f_c}(\dot{\alpha}) = a_{c_2}\dot{\alpha}^2 + a_{c_0}$$

Podemos observar que nuestra expresión cuenta con variables como con derivadas del ángulo de cabeceo hasta su segunda derivada (aceleración angular) y la tensión de cabeceo. Para formar la expresión lineal que explique la dinámica en el eje de cabeceo necesitaremos las derivadas parciales respecto a todas estas variables, esto se ha realizado con el siguiente código en Matlab.

```
 % Pitch
par_pitch_d2alpha   =   double(subs(diff(eq_pitch_NL,   p_d2alpha),   [p_dalpha
p_d2alpha p_alpha p_vc], [0 0 alpha_0 vc_0]));
par_pitch_dalpha = double(subs(diff(eq_pitch_NL, p_dalpha), [p_dalpha p_d2alpha
p_alpha p_vc], [0 0 alpha_0 vc_0]));
par_pitch_alpha  = double(subs(diff(eq_pitch_NL,  p_alpha),  [p_dalpha  p_d2alpha
p_alpha p_vc], [0 0 alpha_0 vc_0]));
par_pitch_vc= double(subs(diff(eq_pitch_NL, p_vc), [p_dalpha p_d2alpha p_alpha
p_vc], [0 0 alpha_0 vc_0]));


eq_pitch_L     =     par_pitch_vc*p_vc     +     par_pitch_alpha*p_alpha     +
par_pitch_dalpha*p_dalpha + par_pitch_d2alpha*p_d2alpha;
```

Este código da como resultado la siguiente expresión linealizada con los valores numéricos correspondientes a las derivadas parciales.

$$\alpha_{lineal} = 0.0055\ \Delta v_c - 0.0227\ \Delta\alpha - 0.0003345\ \Delta\dot{\alpha} - 0.000135\ \Delta\ddot{\alpha}$$

- En el plano horizontal o de giro (Yaw).

$$J_g\ddot{\theta} = r_2 \cdot F_{P_g} \cdot \cos(\alpha) - K_{f_g}\dot{\theta}$$

Donde:

$$F_{P_g}(v_g) = p_{g_1}v_g + p_{c_0} \quad \text{y} \quad K_{f_g}(\dot{\theta}) = a_{g_0}$$

En este caso podemos observar que la expresión cuenta con derivadas del ángulo de giro hasta su segunda derivada y, además, aparece el ángulo de cabeceo y la tensión aplicada al motor de cola. Tendremos que calcular las derivadas parciales de todas estas variables, análogamente al caso del cabeceo, se ha hecho con el siguiente código en Matlab.

```
%Yaw
par_yaw_d2theta = double(subs(diff(eq_yaw_NL, p_d2theta), [p_dtheta p_d2theta
p_theta p_alpha p_vg], [0 0 theta_0 alpha_0 vg_0]));
par_yaw_dtheta = double(subs(diff(eq_yaw_NL, p_dtheta), [p_dtheta p_d2theta
p_theta p_alpha p_vg], [0 0 theta_0 alpha_0 vg_0]));
par_yaw_theta = double(subs(diff(eq_yaw_NL, p_theta), [p_dtheta p_d2theta
p_theta p_alpha p_vg], [0 0 theta_0 alpha_0 vg_0]));
par_yaw_alpha = double(subs(diff(eq_yaw_NL, p_alpha), [p_dtheta p_d2theta
p_theta p_alpha p_vg], [0 0 theta_0 alpha_0 vg_0]));
par_yaw_vg = double(subs(diff(eq_yaw_NL, p_vg), [p_dtheta p_d2theta p_theta
p_alpha p_vg], [0 0 theta_0 alpha_0 vg_0]));


eq_yaw_L = par_yaw_d2theta*p_d2theta + par_yaw_dtheta*p_dtheta +
par_yaw_theta*p_theta + par_yaw_alpha*p_alpha + par_yaw_vg*p_vg;
```

Este código da como resultado la siguiente expresión linealizada con los valores numéricos correspondientes a las derivadas parciales.

$$\theta_{\text{lineal}} = -0.000289\,\Delta v_g - 0.000108\,\Delta\dot{\theta} - 0.000188\,\Delta\ddot{\theta}$$

2.2.3. Modelo en espacio de estados

Gracias a estas ecuaciones que definen nuestras variables de estado, será posible establecer las entradas y salidas de nuestro sistema.

- Variables de estado.

Las variables de estado de nuestro sistema estarán constituidas por:

- El ángulo de cabeceo o pitch (α) y su derivada o velocidad angular respecto al eje horizontal ($\dot{\alpha}$).
- El ángulo de giro o yaw (θ) y su derivada o velocidad angular respecto al eje vertical ($\dot{\theta}$).

$$x = \begin{bmatrix} \alpha \\ \dot{\alpha} \\ \theta \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

- Entradas y salidas del sistema.

Las entradas de nuestro sistema están definidas por u_1 y u_2, estas variables hacen referencia a las tensiones aplicadas a los motores de cabeceo y de cola respectivamente. Por otro lado, las salidas estarán formadas por los ángulos que describe el sistema respecto a los ejes horizontal y vertical.

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix};\ y = \begin{bmatrix} \alpha \\ \theta \end{bmatrix}$$

Tras definir las variables de estado es necesario escribir las ecuaciones de estado resultantes, estas son las siguientes:

$$\frac{dx_1}{dt} = x_2$$

$$\frac{dx_2}{dt} = -167.86\,\mathrm{x}_1 - 2.478\,x_2 + 41.0983\,u_1$$

$$\frac{dx_3}{dt} = x_4$$

$$\frac{dx_4}{dt} = -1.7407\,x_4 + 2.6759\,u_2$$

El código utilizado para crear el modelo en espacio de estados es:

```
%% Modelo en Espacio de Estados
A = [0 1 0 0
  par_pitch_alpha/-par_pitch_d2alpha par_pitch_dalpha/-par_pitch_d2alpha 0 0
  0 0 0 1
  0 0 par_yaw_theta/-par_yaw_d2theta par_yaw_dtheta/-par_yaw_d2theta];

B = [0 0
     par_pitch_vc/-par_pitch_d2alpha 0
     0 0
     0 par_yaw_vg/-par_yaw_d2theta];
```

```
C = [1 0 0 0; 0 0 1 0];
D = zeros(2,2);

modeloTRMS = ss(A,B,C,D);

% Discretizamos el modelo
Ts = 0.05;
ssTRMS_d = c2d(ss(A, B, C, D), Ts);
```

Donde el valor de cada matriz para realizar la representación de estado es:

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

```
A =
          x1       x2       x3       x4
 x1        0        1        0        0
 x2   -167.9   -2.478        0        0
 x3        0        0        0        1
 x4        0        0        0   -1.741

B =
          u1       u2
 x1        0        0
 x2     41.1        0
 x3        0        0
 x4        0    2.676

C =
     x1  x2  x3  x4
 y1   1   0   0   0
 y2   0   0   1   0

D =
     u1  u2
 y1   0   0
 y2   0   0
```

Este modelo se ha discretizado con un tiempo de muestreo de 0.05 segundos. Esto nos da unas matrices resultantes:

```
A =
          x1       x2       x3       x4
 x1    0.8055   0.04381        0        0
 x2    -7.353    0.6969        0        0
 x3         0         0        1  0.04789
 x4         0         0        0   0.9166

B =
          u1       u2
 x1  0.04763        0
 x2       1.8        0
 x3         0  0.00325
 x4         0   0.1281

C =
     x1  x2  x3  x4
 y1   1   0   0   0
 y2   0   0   1   0

D =
     u1  u2
 y1   0   0
 y2   0   0
```

2.2.4. Validación del modelo

Para la validación del modelo desarrollado se han utilizado dos archivos simulink. En uno de ellos se ha implementado el modelo no lineal para así, poder introducirle una señal de tensión para cada uno de los motores y poder obtener la dinámica simulada del sistema. Por otro lado, contamos con un segundo archivo simulink en el que mediante el uso de una tarjeta de adquisición de datos, podremos aportar una cierta señal a la maqueta y recolectar los datos propios del sistema real. Para esta verificación se aportará tanto al sistema real como al modelo la misma entrada en voltios con el objetivo de comprobar sus diferentes respuestas.

En el caso del archivo simulink elaborado para la simulación, el modelo se ha introducido mediante el uso de subsistemas para encapsular cada una de las diferentes dinámicas de nuestro proceso. Como podemos observar en la **Figura 10,** se ha optado por un encapsulado para la dinámica referente al cabeceo y otro para la dinámica del giro.

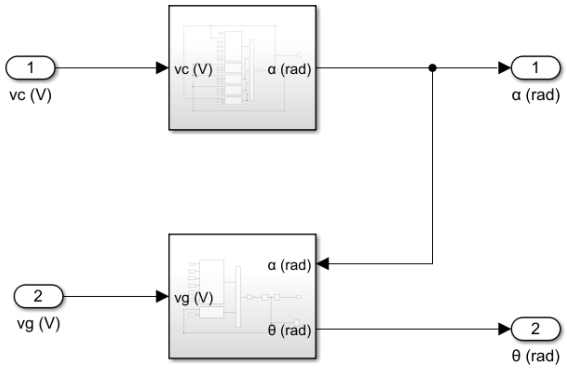

Figura 10. Subsistema principal del modelo no lineal

Cada uno de estos subsistemas cuenta con la implementación de las expresiones propias de los ángulos de giro explicadas anteriormente en este documento. Cada uno de los bloques tiene como entradas las variables que definen ese sistema.

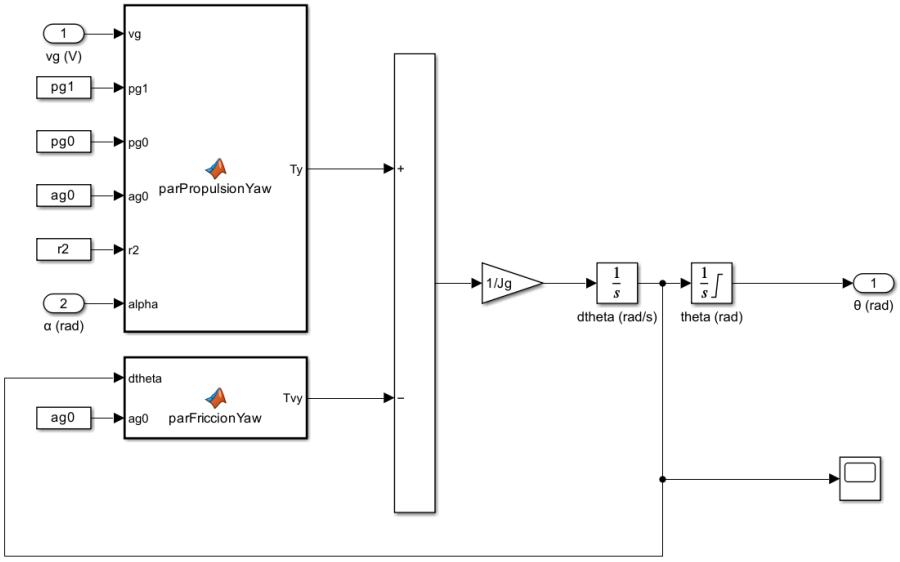

Figura 11. Subsistema referente al giro

En la figura superior podemos ver como los bloques implementados serían el del empuje que aporta el motor de cola y el de la fricción que dificulta este movimiento. Para la obtención del ángulo de giro es necesaria una doble integración.

En el caso del cabeceo, para la obtención del ángulo pitch son necesarios un mayor número de bloques ya que, además de los bloques de empuje del motor y de la fricción propia del movimiento, al ser un movimiento desarrollado en el eje gravitacional contamos con los momentos creados por cada una de las masas que conforman el sistema.

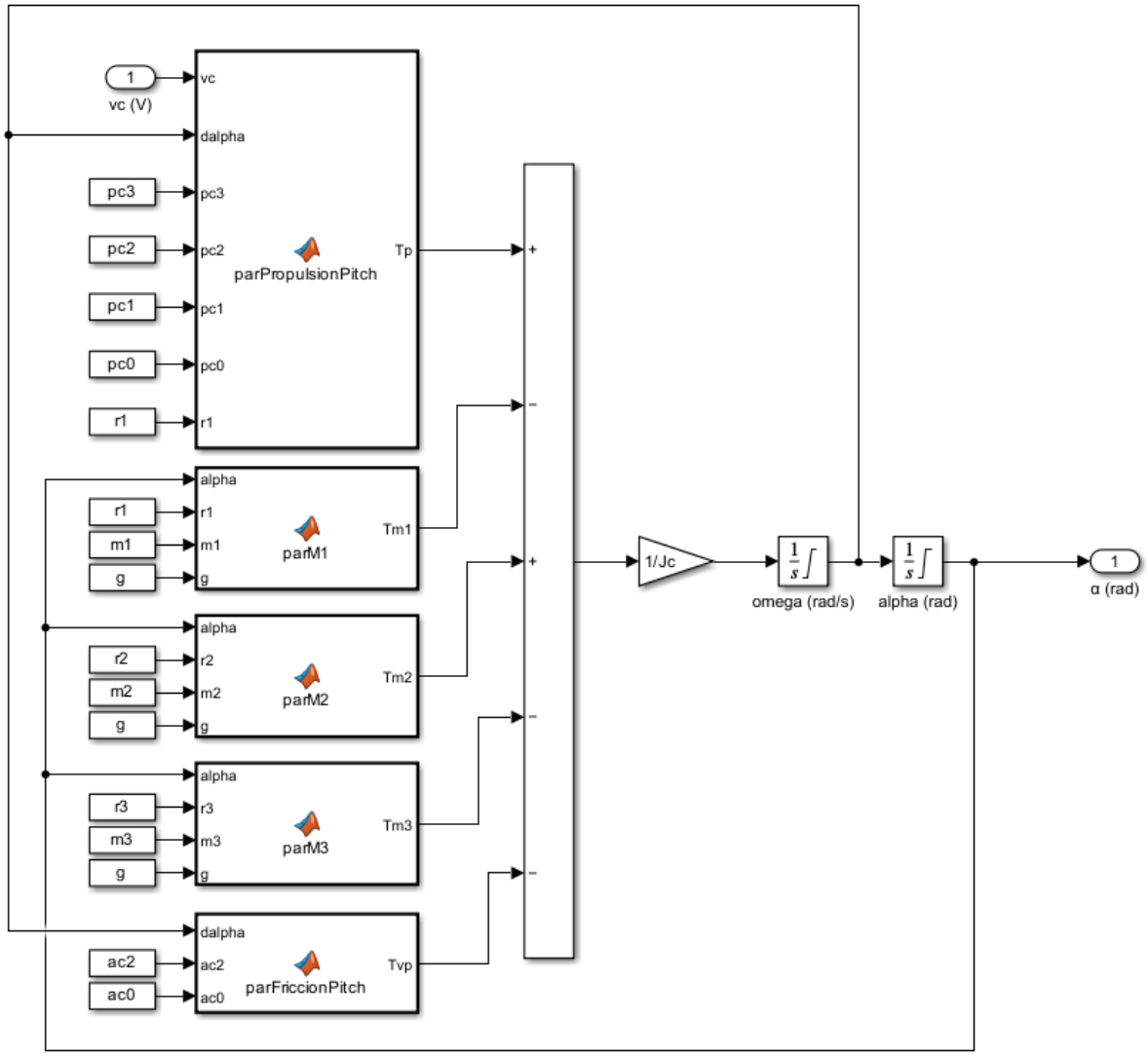

Figura 12. Subsistema referente al cabeceo

En el archivo simulink desarrollado para la recolección de información de la maqueta a utilizar contamos con una serie de bloques que desempeñar una serie de funciones.

- Conexión con la tarjeta de adquisición de datos.

Para esta función contamos con 2 bloques de función que se utilizan para seleccionar el número de canales a utilizar tanto de entradas como de salidas al igual que la configuración de cada una de ellas.

- En el caso del bloque “Analog Input” la configuración seleccionada ha sido la de trabajar con 2 entradas analógicas, cada una de ellas captará la lectura correspondiente al voltaje de cada uno de los sensores del sistema (Pitch y Yaw). En la configuración de estas señales es importante seleccionar que deben estar referidas a masa (single ended).

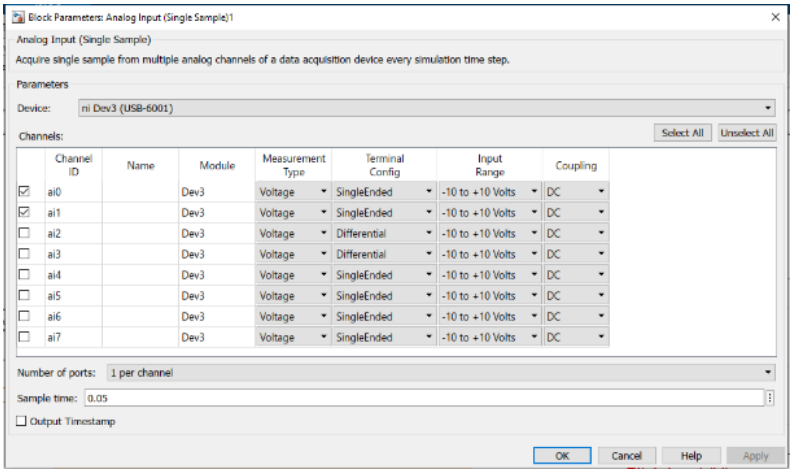

Figura 13. Configuración Entradas Analógicas DAQ

- En el caso del bloque “Analog Output” la configuración seleccionada ha sido la de trabajar con 2 salidas analógicas. El rango de las mismas podría modificarse ya que el rango de trabajo de los motores no es de -10 V a 10V. Cada una de estas salidas se encargará de enviar la correspondiente tensión a cada uno de los motores.

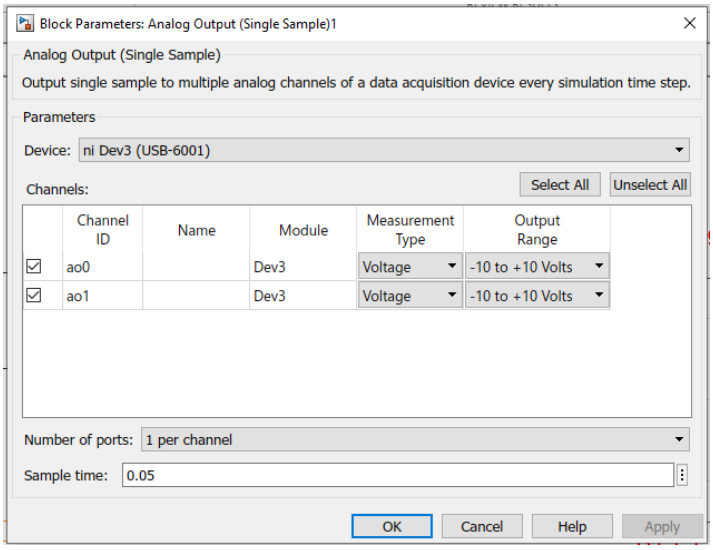

Figura 14. Configuración Salidas Analógicas DAQ

- Descripción del sistema a utilizar.

Estos bloques son aquellos que cuentan con un valor numérico intrínseco a la maqueta de "Twin Rotor" utilizada. Estos bloques son necesarios ya que todas las maquetas no son exactamente iguales y, debido al uso que se hace de ellas no todas responden de la misma forma.

Los bloques que conforman este apartado serían:

- uPitch_0. Tensión necesaria para que la maqueta se encuentre en posición horizontal.

- yPitch_0. Lectura del sensor de cabeceo cuando la maqueta se encuentre en posición horizontal.

- uYaw_0. Tensión necesaria para que la maqueta no se encuentre girando, debe ser 0.

- yYaw_0. Lectura del sensor de giro cuando la maqueta se encuentre en posición de equilibrio.

- Visualización de valores y recolección de estos.

Estos bloques son los diferentes scopes y visualizadores que nos muestran los valores instantáneos de la lectura de los sensores. Además, contamos con un bloque To-Workspace que nos permite guardarnos la información recolectada.

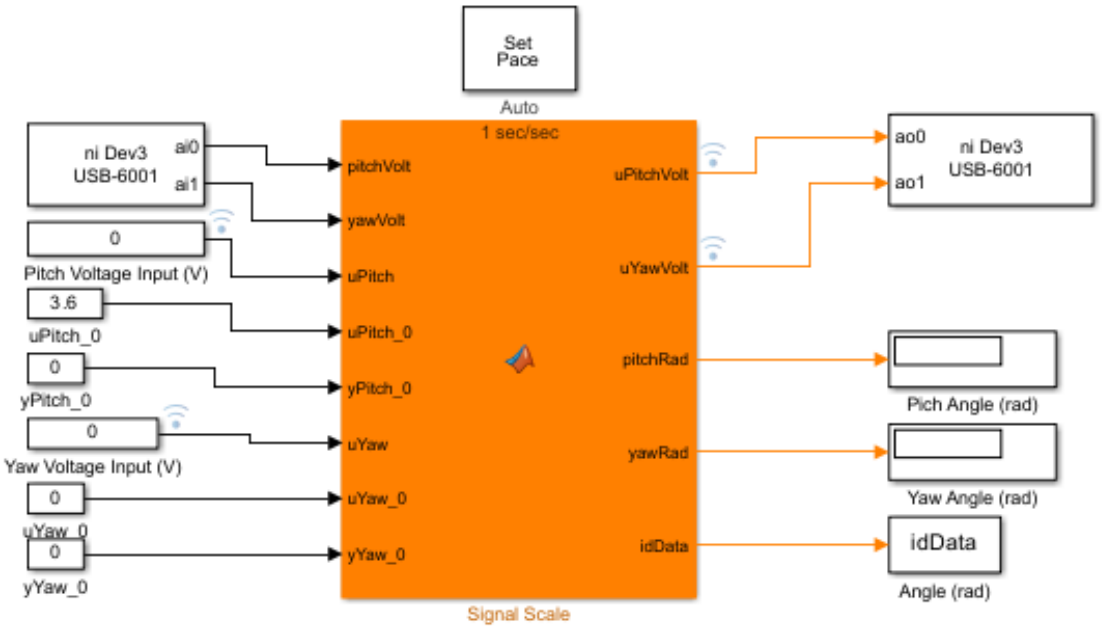

Figura 15. Simulink utilizado para recoger información de la maqueta

Una vez que contamos con los medios suficientes para la comparación de ambos modelos, se ha diseñado el modelo Simulink que podemos observar en la imagen inferior (**Figura 16**). En este archivo podremos comparar tanto el modelo no lineal desarrollado con el modelo lineal y, además, podremos comparar estos con el modelo real tras almacenar la información de la simulación en un vector temporal.

Para que la comparación tenga sentido, se deberá aplicar los mismos cambios de tensión a todos los modelos. Con este objetivo se ha creado un vector que contiene los cambios de referencia llamado

"Vc", que hace referencia a la tensión de cabeceo. Cabe destacar que únicamente tiene sentido comparar el movimiento de cabeceo ya que el giro del sistema se comporta como un integrador y sin aplicar ningún tipo de control, se va a su límite superior o inferior.

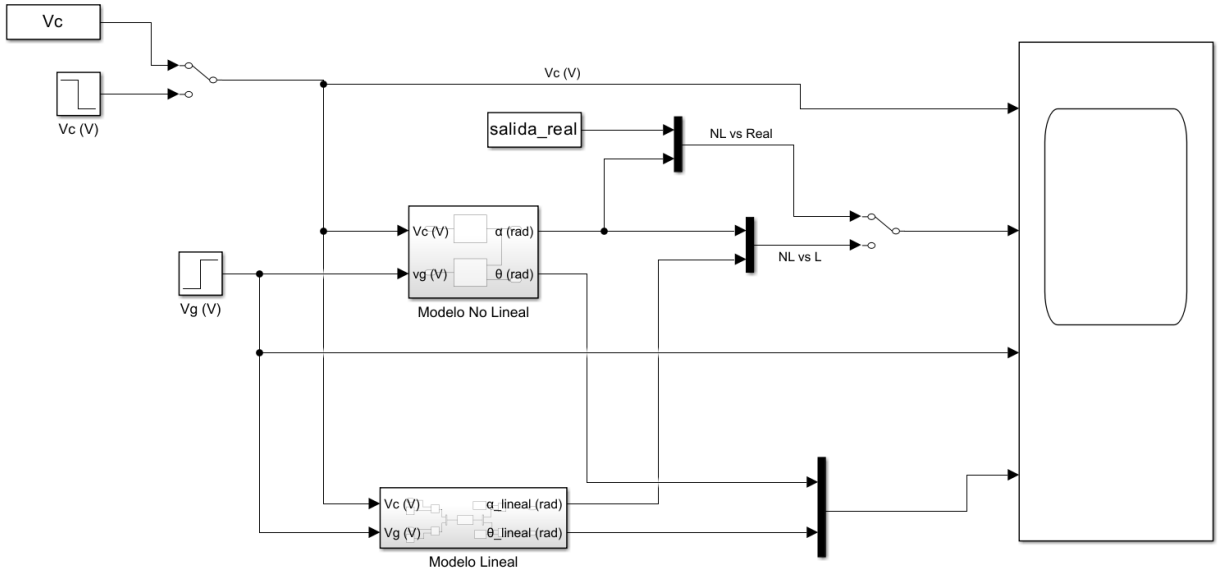

Figura 16. Comparación modelo NL vs modelo L vs Real

De la gráfica sacada de este modelo (**Figura 17**) podemos observar que la tensión necesaria para mantener el punto de equilibrio en la maqueta de helicóptero, en el modelo no lineal ya hace que haya una diferencia en el ángulo de cabeceo. Tras ver los diferentes escalones aplicados, podemos concluir que el motor del cabeceo de nuestra maqueta no cuenta con suficiente potencia para alcanzar ángulos tan elevados. El modelado se da por bueno debido a que el comportamiento del modelo y el sistema real es el mismo para tensiones en las que el proceso real no esté capado debido a la tensión necesaria para llegar al equilibrio. Esta tensión de equilibrio depende de la maqueta utilizada ya que debido a su uso se han degradado y el modelo describe el modelo real sobre el que se partiría al tener un elemento recién fabricado y nosotros trabajaremos con un sistema que ya lleva un uso considerable.

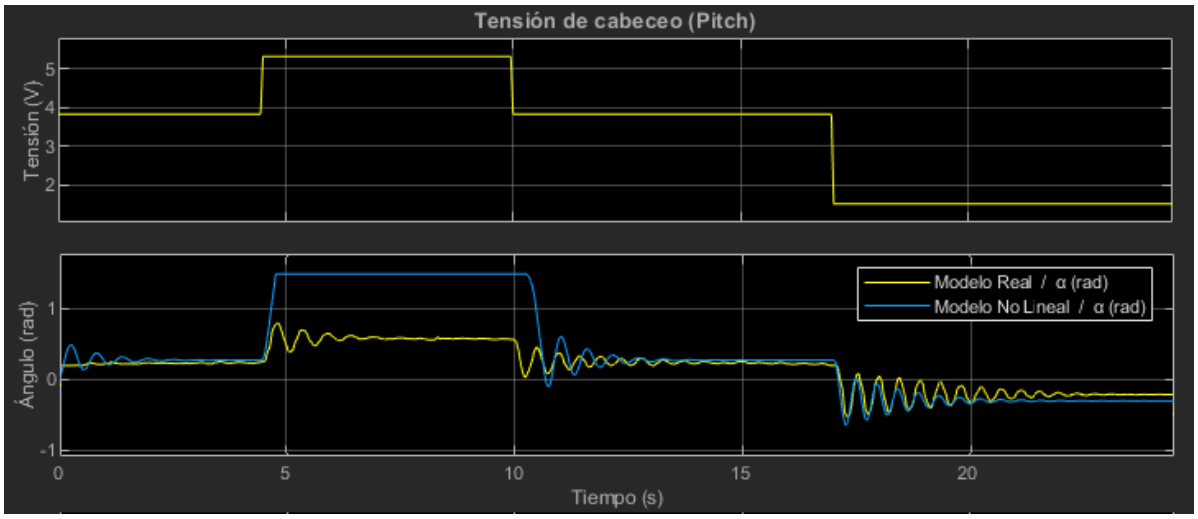

Figura 17. Comparación sistema real vs modelo No Lineal

Capítulo 3

Implementación del MPC

3.1. Introducción al MPC

Tras el correcto modelado de nuestro sistema, se procede a la implementación del control predictivo basado en modelo. Este tipo de control resuelve para cada instante de muestreo, haciendo uso del modelo del proceso, un problema de optimización para obtener una acción de control óptima para el estado del sistema en ese instante.

Esta estrategia hace unos años era inviable debido a la capacidad computacional de la época. Hoy en día es posible realizar este tipo de cálculos incluso para sistemas con dinámicas rápidas ya que el hardware actual permite realizar estos cálculos cada ciertos milisegundos.

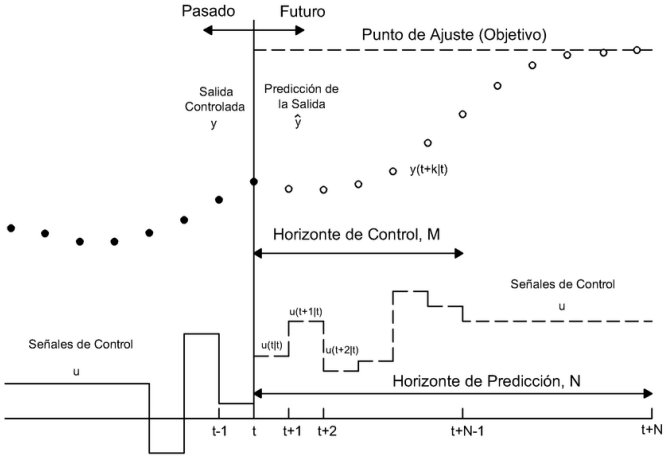

Figura 18. Estrategia MPC

La explicación de la estrategia seguida por todos los controladores basados en modelo es **[7]** :

1. Las salidas futuras para un horizonte determinado N, llamado horizonte de predicción, se predicen en cada instante t utilizando el modelo del proceso. Estas salidas predichas $y(t+k \mid t)$ para $k = 1 \dots N$ dependen de los valores conocidos hasta el instante t (entradas y salidas pasadas) y de las señales de control futuras $u(t+k \mid t)$, $k = 1 \dots N-1$, que son aquellas que se enviarán al sistema y se calcularán.

2. El conjunto de señales de control futuras se calcula optimizando un criterio determinado para mantener el proceso lo más cerca posible de la trayectoria de referencia (que puede ser el punto de consigna en sí mismo o una aproximación cercana de este). Este criterio generalmente toma la forma de una función cuadrática de los errores entre la señal de salida predicha y la trayectoria de referencia predicha. El esfuerzo de control se incluye en la función objetivo en la mayoría de los casos. Se puede obtener una solución explícita si el criterio es cuadrático, el modelo es lineal y no hay restricciones; de lo contrario, se debe

utilizar un método de optimización iterativo. En algunos casos, también se hacen suposiciones sobre la estructura de la ley de control futura, como que será constante a partir de un momento dado.

3. La señal de control $u(t \mid t)$ se envía al proceso mientras que las siguientes señales de control calculadas se descartan, porque en el siguiente instante de muestreo $y(t+1)$ ya será conocida y se deberá repetir el paso 1 con este nuevo valor y todas las secuencias se actualizan. Así, se calcula $u(t+1 \mid t+1)$ (que en principio será diferente a $u(t+1 \mid t)$ debido a la nueva información disponible) utilizando el concepto de horizonte recesivo.

Este control predictivo cuenta con una serie de parámetros modificables que hacen el proceso más o menos efectivo.

- Horizonte de predicción (p). Es el número de instantes futuros que se tienen en cuenta a la hora de calcular las acciones de control. Debe cubrir las dinámicas más representativas del sistema.

- Horizonte de control (c). Es la cantidad de acciones de control que son consideradas para las predicciones. Cuanto más pequeño sea su valor, los cálculos a realizar serán menores. Por otro lado, cuanto mayor sea, mejores serán las predicciones proporcionadas a costa de aumentar la complejidad computacional.

- Tiempo de muestreo (Ts). Es el periodo para el que se ha discretizado el modelo del sistema y, por lo tanto, el controlador deberá ejecutar el algoritmo de control. Este valor no debe ser ni muy elevado para poder contar con un rechazo aceptable de perturbaciones ni demasiado pequeño ya que supone una carga computacional.

El funcionamiento del control predictivo basado en modelo se ve regido por el siguiente diagrama de bloques.

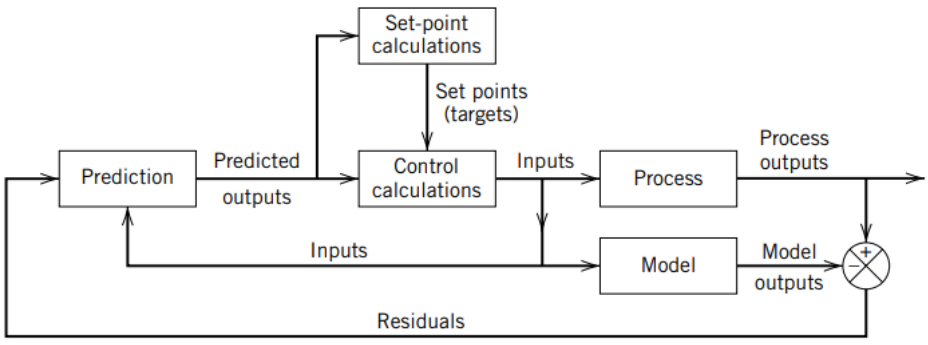

Figura 19. Diagrama de bloques MPC. Fuente: [11]

3.2. Implementación y optimización del control en Matlab

Ya que anteriormente se han obtenido tanto las ecuaciones que describen la dinámica como el modelo del sistema a controlar linealizado alrededor del punto de equilibrio, tras la implementación estas ecuaciones en Matlab se ha logrado obtener el modelo del sistema en espacios de estados y el sistema observador que nos permitirá la estimación de los estados.

La obtención del modelo del proceso es uno de los bloques críticos para la implementación de nuestro control ya que las matrices que describen al mismo serán claves para los cálculos internos.

En este caso, se utiliza el algoritmo DMC (Dynamic Matrix Control) para optimizar las variables de control y obtener un mejor rendimiento en términos de control. La implementación de este código se puede dividir en 2 códigos, por un lado tendríamos el código de inicialización o los cálculos offline en el que se montan las matrices más relevantes durante el control y, por otro lado, tendríamos el control propiamente o cálculos online ya que se realizarán en cada instante de control (Ts).

3.2.1. Cálculos Offline

En este código contamos con la asignación de valores para el número de predicciones y la cantidad de acciones de control a calcular. A su vez, también tendremos que seleccionar los pesos para limitar tanto el error aceptable como la acción de control permisible.

```
%% Inicialización del control SS-MPC
% Cargamos modelo lineal y observador
Practicas_modificado
Crea_Ref

%vc_0 = 3.6; % helicoptero 1

%% Calculos off-line
c = 10;
p = c + 60;

% 1° para cabeceo y 2° para rotación
alfa = [0.5; 1.1];  % relacionado con el error
lambda = [80; 200]; % relacionado con u
```

Una vez que estas variables han sido definidas, se crearán las matrices intrínsecas al modelo necesarias para realizar el control. Ya que la construcción de estas variables es compleja, se han creado varias funciones que realicen este cometido. En el esquema inferior (**Figura 20**) podemos observar las diferentes matrices a formar y la relación existente entre ellas. En color azul oscuro se encuentran aquellas que serán necesarias en el algoritmo de control propiamente y, en azul clarito encontramos aquellas matrices que únicamente sirven como paso intermedio.

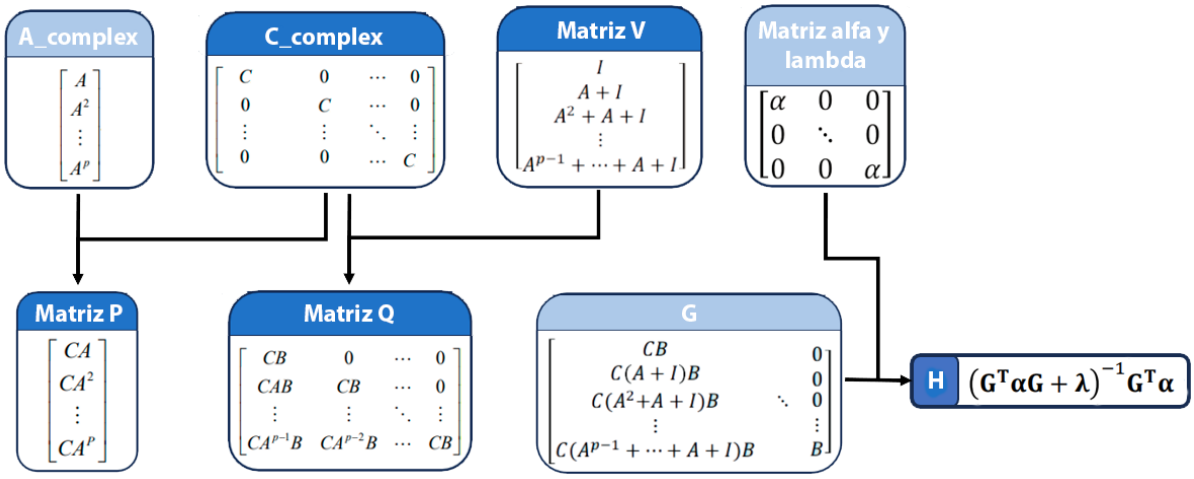

Figura 20. Matrices por construir y su relación

En primer lugar, se calculan tanto la matriz de pesos asignados a los errores en las diferentes referencias como la matriz de pesos relacionada con las acciones de control.

```
alfaM = build_alpha_lambda(p, alfa);
lambdaM = build_alpha_lambda(c, lambda);
```

En la función "build_alpha_lambda" se requiere tanto el vector columna de tantos pesos como sean necesarios para el sistema a controlar como el parámetro "p" o "c" (cuyo significado podemos encontrar en **3.1. Introducción al MPC**) dependiendo de la matriz a formar.

```
function X = build_alpha_lambda(dimension, pesos)

M_X=cell(length(pesos)*dimension,length(pesos)*dimension);
x = 1;

for i = 1:2*dimension
    for j = 1:2*dimension
        if i==j
            M_X{i,i} = pesos(x);
            x = x+1;
        else
            M_X{i,j} = 0;
        end

        if x > length(pesos)
            x = 1;
        end
    end
end

X = cell2mat(M_X);
```

Este código nos devuelve una matriz diagonal en la que los elementos de esta son los pesos proporcionados y su dimensión sería el valor resultante de la multiplicación de la cantidad de pesos por el parámetro "p" o "c".

El cálculo de la estimación de la salida en los "p" instantes siguientes necesita la creación de un número de matrices complejas. La estimación de la salida del sistema está formada por 2 componentes diferentes, la estimación de la salida del sistema sin aplicar ninguna acción de control o, lo que llamaremos respuesta libre, y la acción resultante de las acciones de control aplicadas. Se adjunta la expresión característica para el cálculo de la estimación de la salida del sistema.

$$y_{future_{n\cdot px1}} = y_{free_{n\cdot px1}} + G_{n\cdot pxm\cdot c} \cdot \Delta u_{m\cdot cx1}$$

Donde:

$$y_{free_{n\cdot px1}} = P \cdot x(k) + Q \cdot u(k-1)$$

$$\begin{bmatrix} y(k+1|k) \\ y(k+2|k) \\ \vdots \\ y(k+p|k) \end{bmatrix} = \begin{bmatrix} CA \\ CA^2 \\ \vdots \\ CA^P \end{bmatrix} \cdot x(k) + \begin{bmatrix} CB & 0 & \cdots & 0 \\ CAB & CB & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{p-1}B & CA^{p-2}B & \cdots & CB \end{bmatrix} \begin{bmatrix} u(k|k) \\ u(k+1|k) \\ \vdots \\ u(k+p-1|k) \end{bmatrix}$$

Podemos observar que la expresión de la respuesta libre cuenta con 2 matrices que multiplican a los estados y a la acción de control pasadas.

En primer lugar, se ha calculado la matriz P. Para la creación de esta matriz ha sido necesaria la implementación de una función que nos permita crear una matriz que a su vez contenga la matriz A característica del sistema elevada de 1 a p.

```
A_complex = build_matriz_Acomplex(ssTRMS_d.A,p);
C_complex = kron(eye(p),ssTRMS_d.C);

Matriz_P = C_complex*A_complex;
```

Donde la matriz A_complex se forma con el siguiente código.

```
function [A_complex] = build_matriz_Acomplex(A,p)

A_complex=[];

for i = 1:p
    A_complex=[A_complex; A^i];
end
```

La siguiente matriz que formar sería la matriz Q, esta es la matriz que multiplica a las acciones de control futuras estimadas.

```
  Matriz_V = construct_V(ssTRMS_d.A,p);
  Q = C_complex*Matriz_V*ssTRMS_d.B;
```

Donde la matriz V se forma con el siguiente código.

```
function V = construct_V(A, p)
    V = eye(length(A));
    Aux = eye(length(A));

    for i = 1:p-1
        Aux = Aux + A^i;
        V = [V; Aux];
    end
end
```

Por otro lado, estarían las matrices que nos permiten el cálculo del efecto de las acciones de control a aplicar sobre el sistema. La matriz G tiene la siguiente forma:

```
G = build_G(ssTRMS_d.A,ssTRMS_d.B,ssTRMS_d.C,p,c);
```

```
function G = build_G(A,B,C,p,c)

G=C*B;
I_A=eye(length(A));
[m,~]=size(C);
[~,s]=size(B);

for i=1:p-1
    I_A=I_A+A^i;
    G=[G;C*I_A*B];
end

Aux = G;
for j=1:c-1
    Aux = Aux(1:end-m,:);
    Aux = [zeros(m,s); Aux];
    G = [G Aux];
end
```

Para el posterior cálculo de las acciones de control a aplicar teniendo en cuenta las matrices de pesos es necesaria también la matriz H.

```
H = inv(G'*alfaM*G + lambdaM)*G'*alfaM;
```

Seguidamente, este código también cuenta con la creación de una estructura que aglomera todas las matrices y parámetros necesarios para el funcionamiento del control.

```
%% Variables globales
global MPC;

MPC.p = p;
MPC.c = c;
MPC.Ts = 0.05;

MPC.Gz = ssTRMS_d;
MPC.Obs = SYS_Obs;

MPC.C = C_complex;
MPC.P = Matriz_P;
MPC.V = Matriz_V;
MPC.Q = Q;
MPC.G = G;
MPC.H = H;

% Iniciamos las variables
MPC.u0 = [vc_0; vg_0];
MPC.u_max = [6 6];
MPC.u_min = [0 -6];
MPC.y0 = [alpha_0; theta_0];

MPC.u_ant = [0;0];
MPC.x_ant = [0;0;0;0];
MPC.y_ant = [0;0];
MPC.z_obs_ant = [0;0];
```

3.2.2. Cálculos Online

Este será el algoritmo que se ejecutará cada Ts (valor asignado según el tiempo de discretización seleccionado).

Su objetivo es el de calcular las acciones de control a aplicar sobre el sistema. Para ello, en primer lugar se deberán de recolectar las medidas de los diferentes sensores, para conocer el estado actual, y mediante una estimación del observador, podremos conocer el error entre la realidad y lo estimado. También se calcula tanto la discrepancia de estados, lo cual nos permite llegar al objetivo de control en todo momento al añadirlo en la salida libre.

Una vez que contamos tanto con los objetivos de control como con la salida libre, podremos calcular las acciones de control (“c” acciones de control para cada actuador) que nos permitirá llegar a nuestro objetivo. Ya que se calculan “c” acciones de control y solo podemos aplicar una, deberemos seleccionar las primeras acciones de control calculadas para cada uno de los actuadores.

Al ser todo el cálculo incremental respecto al punto de equilibrio, deberemos sumar al valor obtenido el valor pasado y las tensiones de equilibrio del sistema.

```
% Entradas de la funcion
y_measure = input(1:2) - MPC.y0;
sp_0 = input(3:4) - MPC.y0;

% Estimamos los estados (dalpha dtheta)
z_obs = MPC.Obs.A* MPC.z_obs_ant + MPC.Obs.B*[ MPC.u_ant; MPC.y_ant];
x_obs = MPC.Obs.C*z_obs+ MPC.Obs.D*[ MPC.u_ant; MPC.y_ant];

% Construimos el vector de estados
x_k = [y_measure(1) x_obs(1) y_measure(2) x_obs(2)]';

%% Algoritmo de control
bias = y_measure - MPC.Gz.C*x_k;
bias_ext = [];
sp = [];

for i=1: MPC.p
    bias_ext = [bias_ext; bias]; % Error extendido
    sp = [sp; sp_0]; % Setpoint extendido
end
v = x_k - (MPC.Gz.A* MPC.x_ant + MPC.Gz.B* MPC.u_ant);
y_free = MPC.P*x_k + MPC.Q* MPC.u_ant + bias_ext + MPC.C* MPC.V*v;
du_next = MPC.H*(sp - y_free);
du = [du_next(1); du_next(2)];
u = MPC.u_ant + du + MPC.u0;
```

El último paso de este algoritmo será el de actualizar todas las variables con los valores obtenidos en esta iteración.

```
% Actualizamos las variables para la siguiente iteración
MPC.u_ant = u - MPC.u0;
MPC.x_ant = x_k;
MPC.y_ant = y_measure;
MPC.z_obs_ant = z_obs;
```

3.2.3. Optimización del control

Para la correcta optimización del control MPC se ha desarrollado un script para definir una referencia a seguir en ambos sentidos de movimiento y, así, poder hacer todas las pruebas ante los mismos saltos de referencia. Las referencias desarrolladas tienen la forma mostrada en la siguiente imagen.

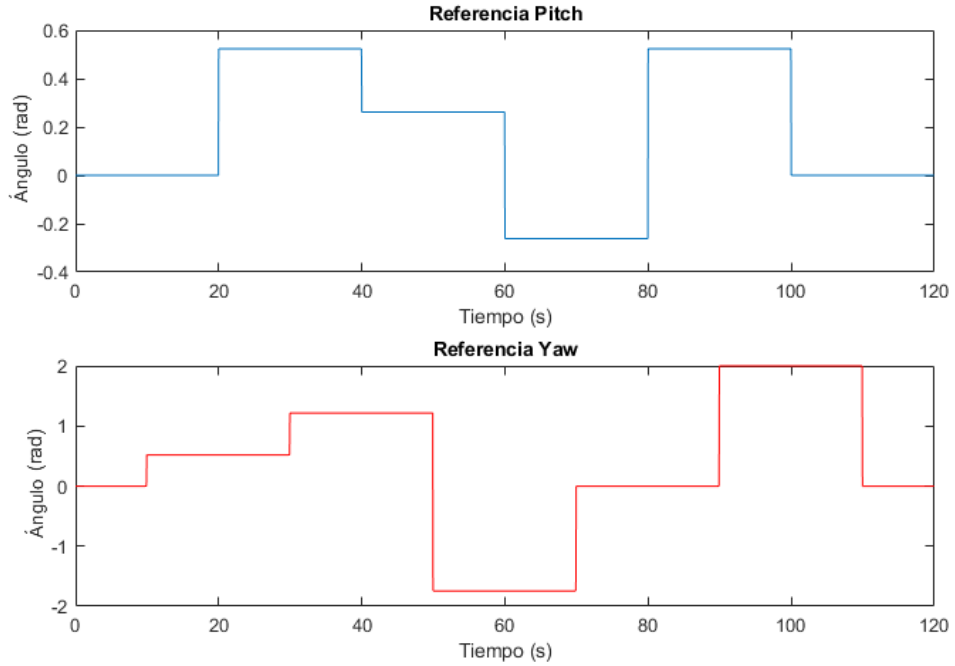

Figura 21. Referencias base

A continuación, se muestra una serie de imágenes que ilustran el comportamiento del sistema frente a cambios arbitrarios en la referencia, tanto para el ángulo de cabeceo como para el de rotación.

Cabe destacar que se han realizado las pruebas de control con el «Helicóptero 1», siendo 3,6 V la tensión para la que el ángulo de cabeceo es cero. Esto se debe a que el motor de esta maqueta es mucho menos potente que el supuesto para la creación del modelo y necesita una mayor tensión para conseguir el empuje suficiente para lograr el punto de equilibrio. En consecuencia, la acción de control real se ve notablemente desfasada respecto de la acción de control simulada. No obstante, la dinámica de ambas señales se asemeja.

En primer lugar, se prueba el helicóptero con unos valores de partida que son $\alpha_1 = 1,\ \alpha_2 = 1,\ \lambda_1 = 100,\ \lambda_2 = 100$. Además, el valor del horizonte de predicción se fija en $p = c + 45$. Para estos valores iniciales, el sistema real se comporta con una inestabilidad notoria en ambos ángulos, como se puede ver en la **Figura 22**.

Se puede observar que esta inestabilidad surge en el momento en el que el ángulo de cabeceo llega a la referencia y ahí comienza a inestabilizarse provocando que, a su vez, el control del ángulo de giro se vea afectado.

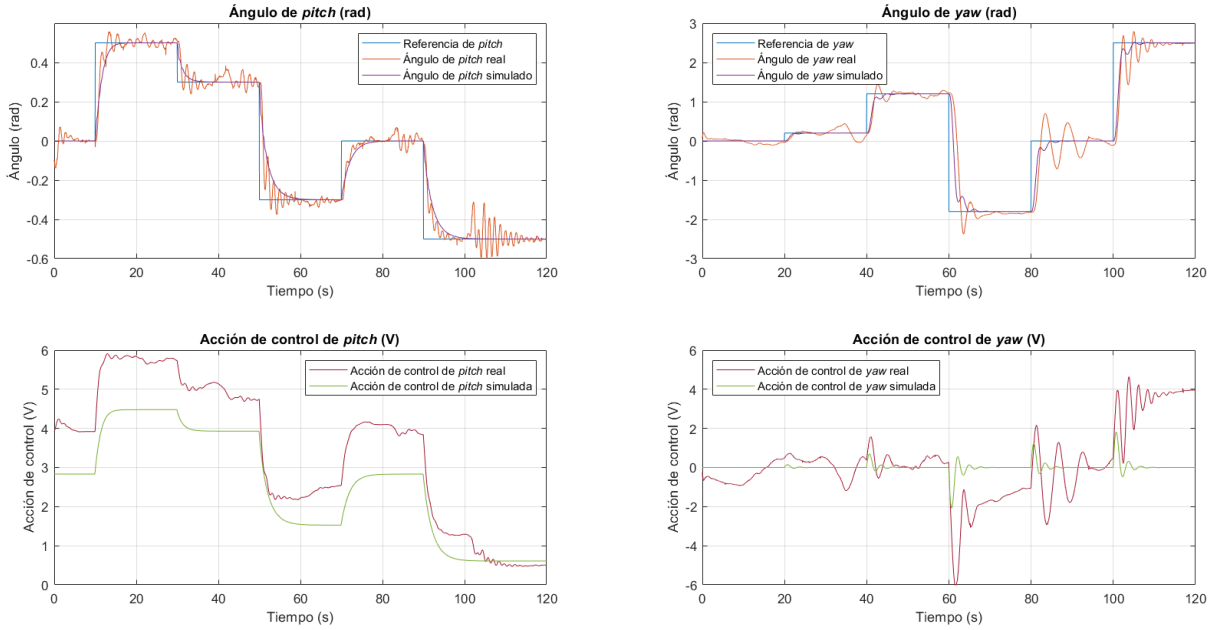

Figura 22. Caso p = c + 45, $\alpha_1 = 1,\ \alpha_2 = 1,\ \lambda_1 = 100,\ \lambda_2 = 100$

Tras conocer la dinámica del sistema con los valores predeterminados, se procede a evaluar el comportamiento del sistema con un cambio en los pesos que, como se observa en la **Figura 23**, provoca que el error en el ángulo de cabeceo o pitch pierda importancia respecto del caso anterior. Esto ha permitido eliminar la inestabilidad resultante de la configuración anterior pero se puede comprobar que, al disminuir este factor α_1, la dinámica de la señal de cabeceo es tan lenta que no es posible alcanzar la referencia (el error no se reduce tan rápidamente). Respecto al ángulo de giro, podemos comprobar que tiene cierta sobreoscilación que sería conveniente eliminar pero llega perfectamente a la referencia.

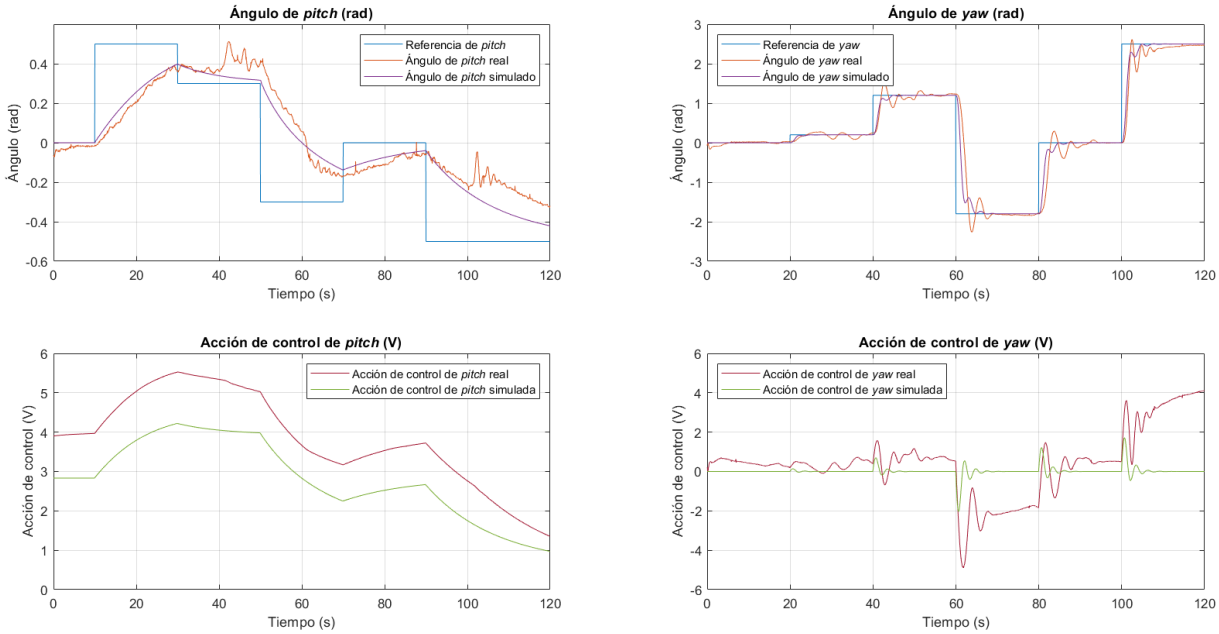

Figura 23. Caso p = c + 45, $\alpha_1 = 0.1,\ \alpha_2 = 1,\ \lambda_1 = 100,\ \lambda_2 = 100$

Para intentar mejorar el control obtenido en el caso anterior, se disminuye el peso que limita el uso de la acción de control correspondiente al motor de cabeceo. Con esta modificación, se consigue alcanzar la referencia de forma rápida y con una dinámica oscilatoria menor que la que se percibe en el primer caso, pero se siguen obteniendo unos resultados claramente indeseables que se muestran en la **Figura 24**.

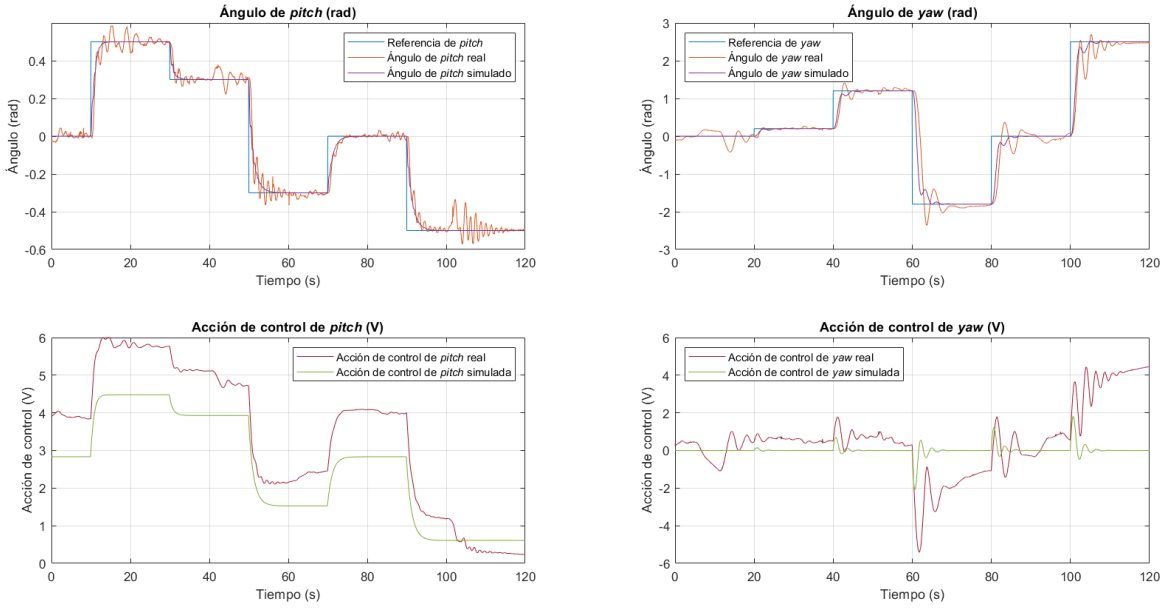

Figura 24. Caso p = c + 45, $\alpha_1 = 0.1,\ \alpha_2 = 1,\ \lambda_1 = 5,\ \lambda_2 = 100$

Después de conseguir un funcionamiento satisfactorio en el eje de cabeceo (o pitch) al aumentar el peso asociado a la acción de control, se inicia con el ajuste del eje de rotación (o yaw). Los parámetros relacionados con este movimiento se fijan en $\alpha_2 = 0.5\ y\ \lambda_2 = 140$ para esta iteración, consiguiendo así un funcionamiento bastante similar al de la simulación, tal y como se ve en la **Figura 25**.

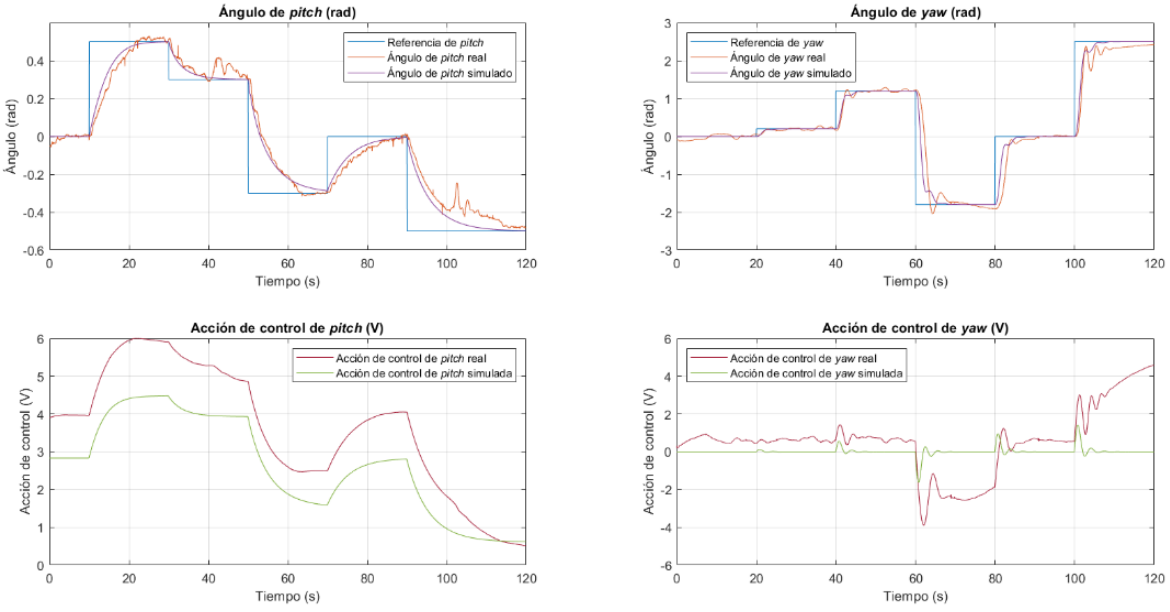

Figura 25. Caso p = c + 45, $\alpha_1 = 0.1,\ \alpha_2 = 0.5,\ \lambda_1 = 30,\ \lambda_2 = 140$

Tras haber explorado las posibilidades que ofrecía la configuración previa, se opta por incrementar el valor del horizonte de predicción, p, en 15 instantes. Con ello, se busca comprobar el impacto de esta variable en el control del sistema real.

Así pues, para unos valores de $\alpha_1 = 0.2,\ \alpha_2 = 0.9\ y\ \lambda_1 = 150,\ \lambda_2 = 200$, se puede observar en la **Figura 26** que la respuesta en el eje de cabeceo tiene unos pesos asignados inadecuados, ya que nuevamente no se consigue llegar a la referencia. Este problema se puede solventar aumentando el peso asociado al error, α_1, o disminuyendo el peso asociado a la utilización de la acción de control, λ_1.

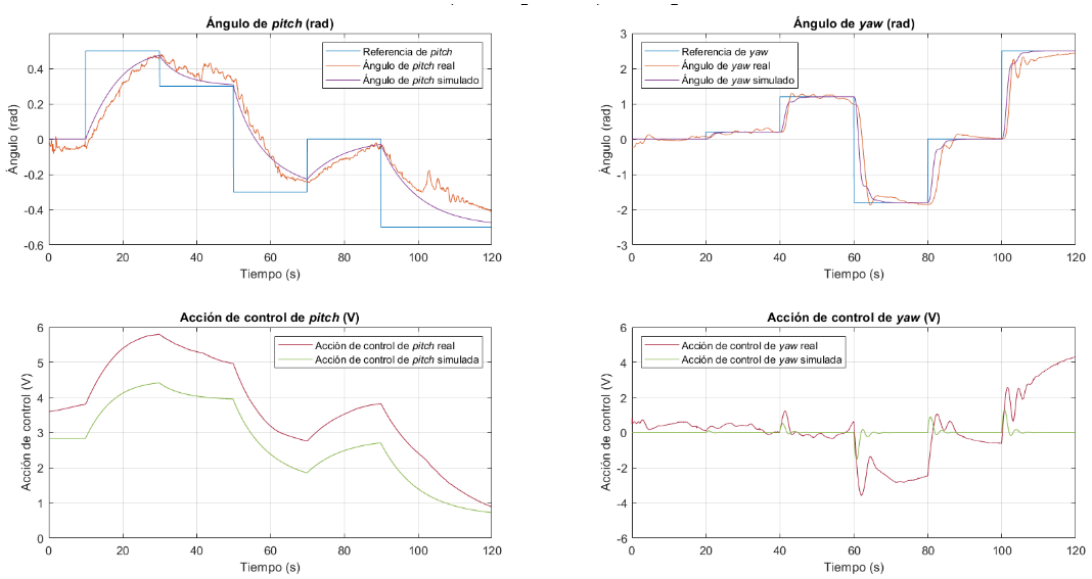

Figura 26. . Caso p = c + 60, $\alpha_1 = 0.2,\ \alpha_2 = 0.9,\ \lambda_1 = 150,\ \lambda_2 = 200$

Realizando los cambios comentados anteriormente, se logra llegar a la respuesta que se muestra en la **Figura 27**. En este caso, se logra alcanzar ambas referencias de forma rápida y sin oscilaciones excesivas.

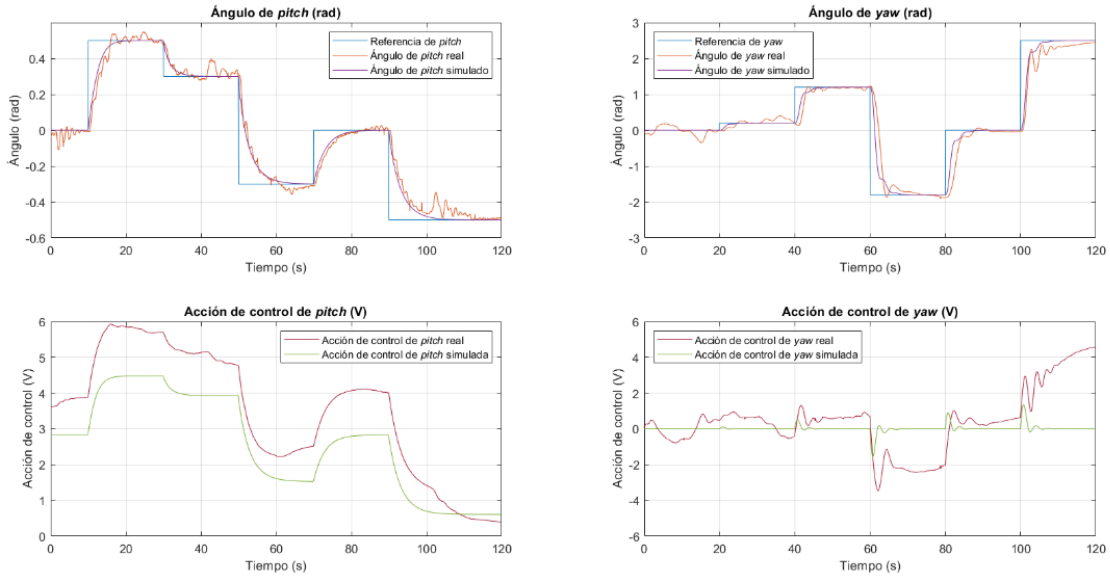

Figura 27. Caso p = c + 60, $\alpha_1 = 0.5,\ \alpha_2 = 0.9,\ \lambda_1 = 100,\ \lambda_2 = 200$

Otro ejemplo con una dinámica muy similar a la iteración anterior es la ilustrada en la **Figura 28**.

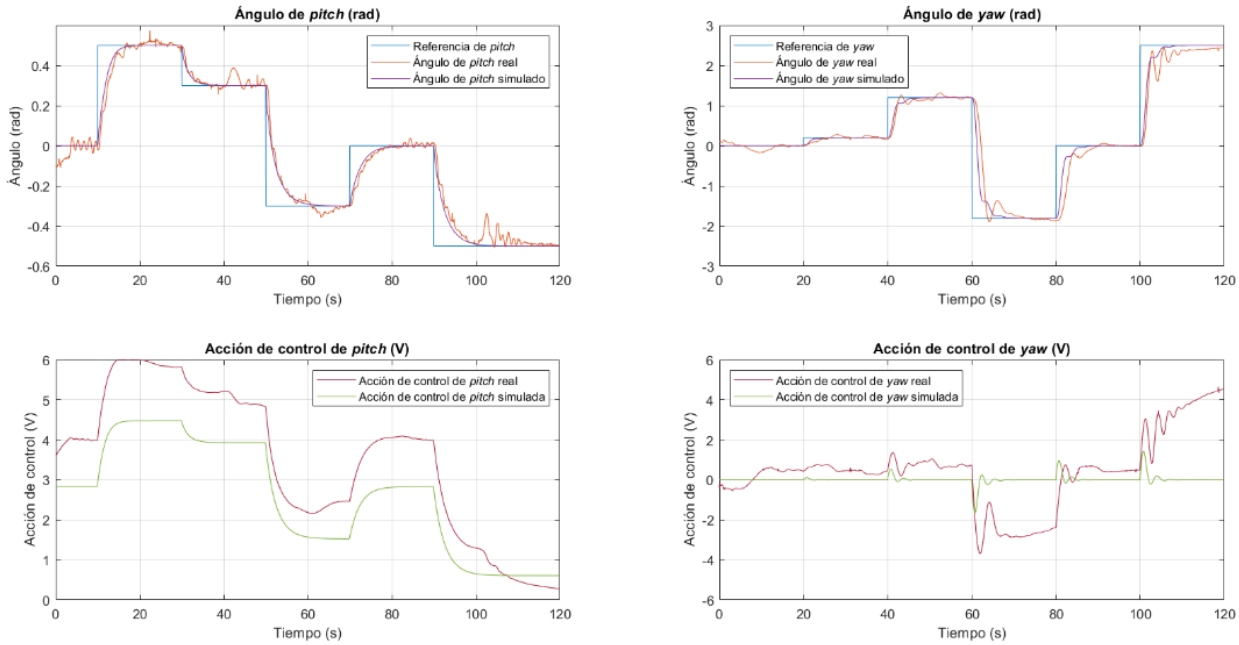

Figura 28. Caso p = c + 60, $\alpha_1 = 0.5,\ \alpha_2 = 1.1,\ \lambda_1 = 80,\ \lambda_2 = 200$

Finalmente, se adjunta una tabla para ver el progreso logrado en el proceso de optimización mediante la modificación de pesos.

Tabla 2. Simulaciones y pruebas reales con distintos valores de α, λ *y* p.

	α_1	α_2	λ_1	λ_2	Observaciones
$p = c + 45$	0.1	0.5	30	140	No se llega a la referencia
	0.1	1	5	100	Hay sobreoscilación
	0.1	1	20	120	Hay sobreoscilación
	0.1	1	20	200	Hay sobreoscilación
	0.1	1	20	250	Comportamiento adecuado
	0.1	1	30	130	No se llega a la referencia Hay sobreoscilación
	0.1	1	100	100	No se llega a la referencia
	0.1	1.2	30	140	No se llega a la referencia Hay sobreoscilación
	0.1	1.5	30	140	No se llega a la referencia Hay sobreoscilación
	0.25	0.95	40	110	Comportamiento adecuado Hay cierta sobreoscilación
	1	1	100	100	Hay mucha sobreoscilación
	1	1	100	300	Hay mucha sobreoscilación
$p = c + 60$	0.2	0.9	150	200	No se llega a la referencia
	0.25	0.9	150	200	No llega a la referencia
	0.5	0.8	140	200	Hay sobreoscilación
	0.5	0.9	100	200	Comportamiento adecuado
	0.5	0.9	140	200	Comportamiento adecuado Llega justo a la referencia
	0.5	1.1	80	200	Comportamiento adecuado Hay cierta sobreoscilación
	0.5	1.1	100	200	Comportamiento adecuado Hay cierta sobreoscilación

En general, se ha podido comprobar que existe una discrepancia entre la dinámica experimentada en el laboratorio y la respuesta obtenida del modelo teórico simulado. Se ha identificado una sobreoscilación excesiva en el cabeceo ante cambios negativos en la referencia. Esto es debido a que el movimiento la hélice coincide en dirección y sentido con la gravedad, provocando una inestabilidad en el sistema. Asimismo, existe cierto acoplamiento entre los movimientos de cabeceo y rotación, comportamiento que no se produce en el sistema simulado.

Además, en cuanto a las acciones de control, al margen del desfase en la acción de control de pitch respecto de la simulación, se puede constatar un incremento significativo en el nivel de tensión necesario para controlar el movimiento de yaw, alcanzando diferencias superiores a los 4 V en los últimos instantes de la simulación, momento en que la acción de control se debería estabilizar por haber alcanzado el régimen permanente en ángulo de yaw.

3.3. Implementación del control en CodeSys

Una vez logrado el correcto funcionamiento del control en Matlab, ya es hora de transferir este control hacia la plataforma final donde se deberá ejecutar. Esta plataforma es el PLC de Phoenix Contact llamado PLCNext, para el desarrollo del control se ha utilizado el software de código libre llamado CoDeSys [9]. La selección de este software sobre el software (PLCNEXT Engineer) diseñado por la compañía propietaria del propio PLC (Phoenix Contact) ha sido debido a la existencia de librerías con funciones que podrían ser útiles para el desarrollo de este proyecto y el hecho de que CoDeSys es compatible con una amplia variedad de PLCs industriales.

Para realizar la migración del código de una plataforma hacia la otra, en primer lugar se contempló la utilización de una aplicación propia de Matlab que permite la creación de código con compatibilidad para una gran cantidad de programas de control. Esta aplicación se llama PLCcoder [8] y cuenta con la implementación de CoDeSys 3.5 como posibilidad para la generación de código. Esta es la versión más actual y, por lo tanto, la seleccionada para la implementación del control en el PLC .

Para seleccionar este tipo de código se deberá abrir la configuración de la aplicación y seleccionarlo, en esta configuración además aparece el directorio donde se guardará el código una vez creado.

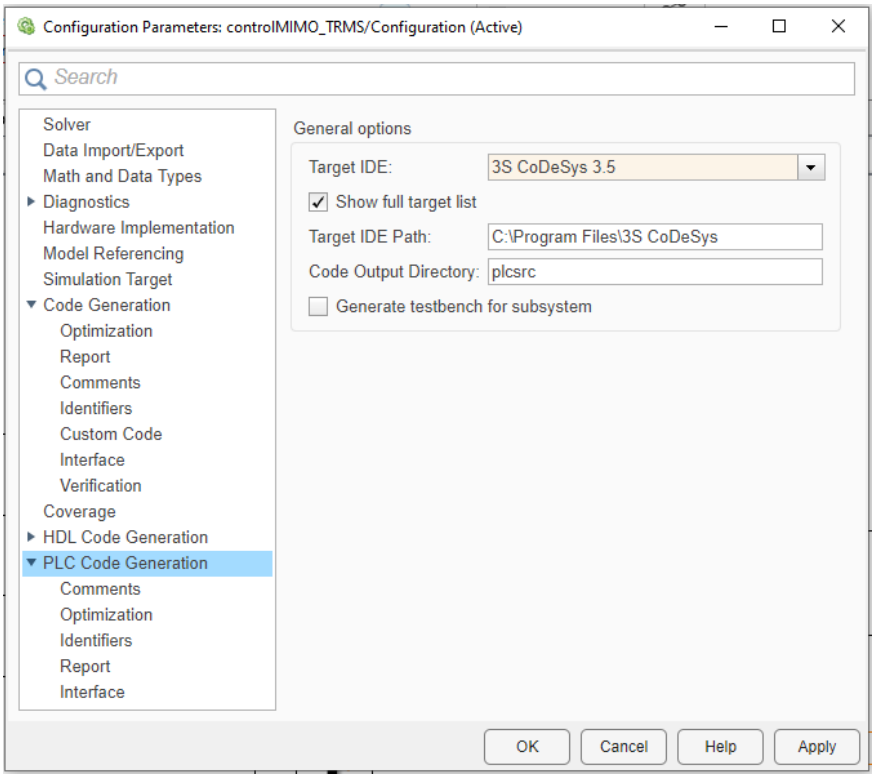

Figura 29. Configuración PLCcoder

La utilización de esta aplicación es muy clara, todo el control debe estar depositado sobre un subsistema de Matlab el cual será convertido al tipo de código seleccionado por el usuario. Una vez que este subsistema está creado, se deberá dar botón derecho sobre este y seleccionar “PLC code”, donde se abrirá una ventana donde se debe habilitar la opción de tratar el subsistema como una unidad atómica y, por último, se deberá pulsar sobre “Generate PLC Code”.

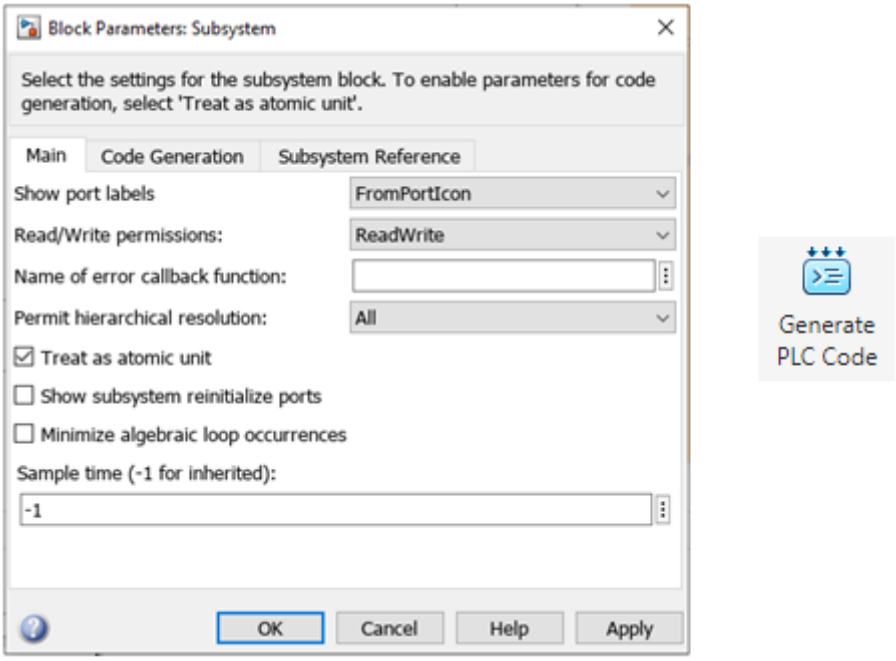

Figura 30. Paso utilización PLCcoder

Cabe destacar que depende del bloque utilizado para la realización de este código, el PLCcoder no será capaz de crear el código pertinente. Ya que para la implementación de este código que se debe ejecutar periódicamente se ha utilizado un bloque de "Interpreted Matlab Function", la aplicación PLCcoder no es capaz de transcribirla en otro lenguaje.

Tras comprobar las limitaciones de esta aplicación, se ha optado por la creación desde cero del código necesario para el control de nuestro sistema. Para la creación de los programas se ha seleccionado texto estructurado debido a su similitud con el código implementado en Matlab y su cómodo desarrollo.

Como hemos visto en el apartado anterior **(3.2. Implementación y optimización del control en Matlab)**, ya sabemos que como mínimo serán necesarios dos scripts o POUs en el lenguaje de CoDeSys, para el desarrollo tanto de los cálculos offline como del cálculo online.

3.3.1. Código offline en CoDeSys.

En este caso lo primero que se ha realizado ha sido la definición de las variables a utilizar. Cabe destacar que CoDeSys no permite de forma nativa el paso a funciones de matrices de dimensiones dinámicas debido a que puede dar problemas al provocar desbordamientos y errores debido a la incorrecta utilización de la memoria del PLC. Por lo tanto, hemos decidido trabajar con dimensiones fijas y, para ello, el trabajo realizado anteriormente en Matlab es de gran importancia al aportarnos el tamaño de cada una de las variables. Si este trabajo no se hubiese realizado con anterioridad, en este momento tocaría calcular las dimensiones de cada matriz, la longitud de cada vector y todo esto teniendo en cuenta que estas dimensiones son dependientes de los parámetros p y c seleccionados propios del control predictivo basado en modelo.
Por lo tanto, se considera indispensable el cálculo preliminar en Matlab o en cualquier otro software que permita dimensiones variables hasta lograr un resultado aceptable para el control propuesto.

El código offline en CoDeSys comienza con la definición de los parámetros de las matrices propias del sistema (A_TRMS, B_TRMS, C_TRMS y D_TRMS) y del observador (A_Obs, B_Obs, C_Obs y D_Obs) elemento a elemento. Además, será necesaria la definición de la matriz identidad (4x4 para el control realizado), los pesos α y λ y el valor de los parámetros p y c.

Una vez que se han definido todas las variables necesarias, se debe continuar construyendo las matrices al igual que en Matlab. En el esquema inferior (**Figura 20**) podemos observar las diferentes matrices a formar y la relación existente entre ellas. En color azul oscuro se encuentran aquellas que serán necesarias en el algoritmo de control propiamente y, en azul clarito encontramos aquellas matrices que únicamente sirven como paso intermedio.

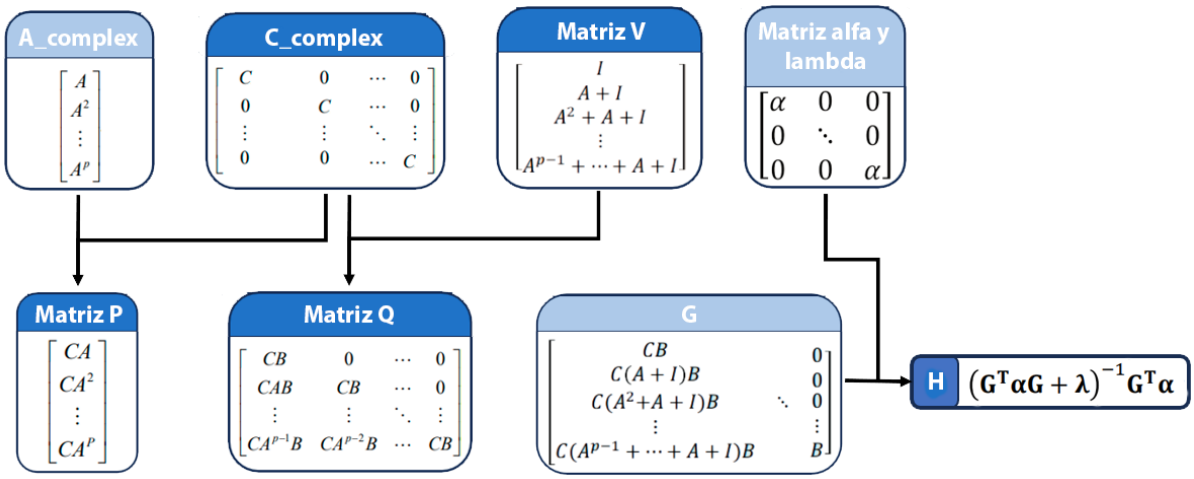

Figura 31. Matrices por construir y su relación

Para las matrices que contienen los pesos α y λ se ha desarrollado un código que permite su construcción para así, poder contar con un código estructurado y segmentado para la detección de errores. A la hora de conectar con el PLC y realizar el cálculo se produce una excepción que no ha permitido el cálculo de las mismas de esta forma, cabe destacar que si realizamos el cálculo seleccionando la opción de hacerlo en simulación (sin necesidad de un PLC real), esta excepción no se produce y realiza el cálculo correctamente. La solución aportada ha sido la transferencia de este código a la función "madre", lo que produce que crezca en magnitud pero no se produce dicha excepción y el cálculo es correcto.

```
//Matrices Alpha y Lambda

i := 1;
x := 1;
FOR i := 1 TO UPPER_BOUND(alfa,2) * gvl.p DO
    FOR j := 1 TO UPPER_BOUND(alfa,2) * gvl.p  DO
        IF i = j THEN
            alfaM[i, j] := alfa[1,x];
        ELSE
            alfaM[i, j] := 0;
        END_IF
        x := x + 1;

        IF x > UPPER_BOUND(alfa,2) THEN
            x := 1;
        END_IF
    END_FOR
END_FOR
```

Tras la construcción de estas matrices y la necesidad de avanzar en el cálculo de matrices más complejas, se ha descubierto que CoDeSys no cuenta con la capacidad de realizar cálculos matriciales de forma nativa. Como este tipo de cálculos matriciales son indispensables para la implementación de un control MPC de este tipo, se ha realizado una búsqueda para evitar la programación de estas. Únicamente se ha logrado localizar librerías con las funciones necesarias para realizar operaciones con matrices 3x3 y, ya que nuestro control cuenta con matrices de dimensiones muy superiores, finalmente se ha requerido de la programación de una serie de funciones para lograr que CoDeSys realice todas las operaciones básicas necesarias.

Para la construcción de la matriz "A_complex" ha sido necesario la creación de un código que permita el cálculo de la potencia de una matriz hasta el valor de p seleccionado. Como podemos observar en el código desarrollado, en primer lugar se calcula la matriz elevada y posteriormente se realiza la definición elemento a elemento.

```
FOR elevado := 1 TO gvl.p DO
    A_elevada := MatrixPower(gvl.A_TRMS, elevado);
    FOR i := 1 TO 4 DO
        FOR j := 1 TO 4 DO
            A_complex[4*(elevado-1)+i,j] := A_elevada[i,j];
            // Utiliza la función MatrixPower para elevar la matriz A a la potencia i
        END_FOR
    END_FOR;
END_FOR
Build_A_complex := A_complex;
```

Para el cálculo de la matriz elevada se ha desarrollado el siguiente código en el que es necesario pasar como variables tanto la matriz a elevar como el índice al que es necesario elevarla. Además, como el método implementado para elevar las matrices es una multiplicación recursiva, también se ha tenido que contar con la programación de una función que permita la multiplicación de variables.

```
IF p = 0 THEN
    // Caso especial: A elevado a la potencia 0 es la matriz identidad
    FOR k := 1 TO 4 DO
        result[k, k] := 1.0;
        // Establece la diagonal principal a 1
    END_FOR
ELSE
    result := A;
    // Inicializa el resultado con la matriz de entrada A

    FOR k := 2 TO p DO
        result := Multiplica_Matrices_4x4(result, A);
        // Multiplica el resultado actual por la matriz A en cada iteración
    END_FOR
END_IF

MatrixPower := result;
```

Como se ha comentado anteriormente, CoDeSys no cuenta de forma nativa con ningún tipo de cálculo matricial ni permite la utilización de matrices de dimensiones variables. Por lo tanto, a la hora de multiplicar se ha desarrollado una función escalable para cualquier dimensión en la que simplemente se deba modificar el tamaño de la matriz resultante. Se adjunta el código implementado para el cálculo de multiplicaciones de matrices en el que la matriz resultante es una 4x4.

```
o := UPPER_BOUND(matrixA,2);
p := UPPER_BOUND(matrixB,1);

IF o <> p THEN
   (* Las matrices no son compatibles para multiplicación *)
   RETURN;
END_IF;

FOR m := LOWER_BOUND(matrixA,1) TO UPPER_BOUND(matrixA,1) DO
    FOR n := LOWER_BOUND(matrixB,2) TO UPPER_BOUND(matrixB,2) DO
        resultMatrix[m, n] := 0;
        FOR k := LOWER_BOUND(matrixA,2) TO UPPER_BOUND(matrixA,2) DO
            resultMatrix[m, n] := resultMatrix[m, n] + (matrixA[m, k] * MatrixB[k, n]);
        END_FOR;
    END_FOR;
END_FOR;
Multiplica_Matrices_4x4 := resultMatrix;
```

Como se puede observar, en primer lugar lo que se realiza es una comprobación de que la operación es realizable (el número de columnas de la primera matriz debe ser igual que el número de filas de la segunda matriz).

Para el cálculo de la matriz C_complex en la que en Matlab únicamente era necesario el uso del siguiente comando (kron(eye(p),ssTRMS_d.C)) el cual crea una matriz identidad del tamaño de la primera variable que se pasa a la función (pxp en este caso) y, en segundo lugar, rellena las posiciones que estén rellenas con 1 por la segunda variable pasada. En la implementación en CoDeSys ha sido necesario el siguiente código:

```
FOR k := 1 TO gvl.p DO
    i := 1;
    FOR i := 1 TO UPPER_BOUND(gvl.C_TRMS,1) DO
        j := 1;
        n := 1;
        FOR j := 1 TO UPPER_BOUND(gvl.C_TRMS,1)*gvl.p DO
            IF (UPPER_BOUND(gvl.C_TRMS,2)*(k-1)+1 <= j AND j <= UPPER_BOUND(gvl.C_TRMS,2)*k) THEN
                C_complex[i+2*(k-1), j] := gvl.C_TRMS[i, n];
                // Asigna los elementos de la matriz base
                n:=n+1;
            ELSE
                C_complex[i+UPPER_BOUND(gvl.C_TRMS,1)*(k-1), j] := 0;
                // Rellena con 0 las posiciones fuera de la matriz base
            END_IF
        END_FOR
    END_FOR
END_FOR
```

Tras la obtención de las matrices C_complex y A_complex, el cálculo de la matriz P se realiza utilizando una función de multiplicación con la característica específica de la dimensión de dicha matriz.

```
Matriz_P := Multiplica_Matrices_140x4(C_complex,A_complex);
```

Para el cálculo de la matriz V se ha optado por la implementación de una función externa que será llamada pasándole la matriz A_TRMS.

```
// Declaramos la identidad 4x4
    Identidad := gvl.Identity_4x4;

// Inicializamos variables
    i := 1;
    j := 1;
    contador := 0;
    Aux := Identidad;

FOR elevado := 0 TO gvl.p-1 DO
    A_elevada := MatrixPower(gvl.A_TRMS, elevado);
    i := 1;
    IF contador<>0 THEN
        Aux := Suma_Matrices_4x4(Aux,A_elevada);
    END_IF
    FOR i := 1 TO 4 DO
        j := 1;
        FOR j := 1 TO 4 DO
            IF contador<>0 THEN
                V[4*elevado+i,j] := Aux[i,j];
            ELSE
                V[i,j] := Identidad[i,j];
            END_IF
        END_FOR
    END_FOR;
    contador := contador + 1;
END_FOR

Build_V := V;
```

Como podemos observar, la construcción de esta matriz se realiza utilizando la función comentada anteriormente que permite elevar las matrices a cierto índice seguido de un comprobador de que si el índice a elevar es distinto de 0, la matriz resultante debe ser la suma de la matriz calculada en el paso anterior y la matriz calculada en ese instante. Para la primera iteración de este cálculo la matriz resultante es la matriz identidad 4x4.

Con las matrices previamente calculadas, la construcción de la matriz Q se realiza llevando a cabo una multiplicación previa y una segunda multiplicación, ya sí, para obtener la matriz Q.

```
Matriz_C_complex_V := Multiplica_Matrices_140x4(C_complex,Matriz_V);
Q := Multiplica_Matrices_140x2(Matriz_C_complex_V,gvl.B_TRMS);
```

La construcción de la matriz G se ha realizado una vez más con una función externa debido a la gran cantidad de código que contiene y, así, el código principal mantiene una estética ordenada y facilita la detección de errores.

```
G := Build_G(gvl.A_TRMS,gvl.B_TRMS,gvl.C_TRMS);
```

Como podemos observar, para el cálculo de la matriz G es necesario tanto la matriz A, como la B y la C del sistema a controlar. La estrategia seguida para su creación ha sido la de, en primer lugar,

conseguir una matriz con los valores de su primera columna (matricial), este primera columna se puede observar en la **Figura 20**.

```
// Declaramos la identidad 4x4
    Identidad:=gvl.Identity_4x4;

// Inicializamos variables
m := UPPER_BOUND(matrix_C,1);
n := UPPER_BOUND(matrix_B,2);
i := 1;
j := 1;
contador := 0;

G := Multiplica_Matrices_2x2(Matrix_C,Matrix_B);
I_A := Identidad;

FOR elevado := 0 TO gvl.p-1 DO
    A_elevada := MatrixPower(gvl.A_TRMS, elevado);
    i := 1;
    IF contador<>0 THEN
        I_A := Suma_Matrices_4x4(I_A,A_elevada);
    END_IF
    FOR i := 1 TO 2 DO
        j := 1;
        FOR j := 1 TO 2 DO
            IF contador<>0 THEN
                C_IA := Multiplica_Matrices_2x4(Matrix_C,I_A);
                G := Multiplica_Matrices_2x2(C_IA,Matrix_B);
                G_fin[2*elevado+i,j] := G[i,j];
            ELSE
                G_fin[i,j] := G[i,j];
            END_IF
        END_FOR
    END_FOR;
    contador := contador + 1;
END_FOR
```

Tras lograr la formación de esta matriz, el siguiente paso para conseguir la matriz G definitiva es el de crear un “offset” de valores nulos que se irá incrementando respecto aumenta el número de columna a rellenar.

```
Aux := G_fin;
fila := UPPER_BOUND(Aux,1);
column := UPPER_BOUND(Aux,2);
k := 1;
FOR k := 1 TO gvl.c DO
    i := 1;
    i_2 := 1;
    FOR i := 1 TO fila DO
        j := column*(k-1)+1;
        FOR j := column*(k-1)+1 TO column*k DO
            IF i >= 1 AND i<= column*(k-1) AND k-1 <> 0 THEN
                result[i,j] := 0;
            ELSE
                IF j MOD 2 = 1 THEN
                    result[i, j] := Aux[i_2, 1];
                ELSE
                    result[i, j] := Aux[i_2, 2];
                END_IF
            END_IF
        END_FOR
        IF NOT (i >= 1 AND i<= fila*(k-1)) THEN
            i_2 := i_2 +1;
        END_IF
    END_FOR
END_FOR

Build_G := result;
```

El cálculo de la matriz H se debe calcular mediante la siguiente expresión:

$$H = (G' * alfaM * G + lambdaM)^{-1} * G' * alfaM$$

Debido a la imposibilidad en el cálculo de la inversa de matrices de dichas dimensiones con métodos matemáticos “simples”, se ha optado por el cálculo en Matlab y la definición de la matriz resultante elemento a elemento en CoDeSys. Pese a dicha imposibilidad, en un primer lugar se programaron los POUs necesarios para el cálculo de matrices inversas de dimensiones 4x4. Los POUs necesarios para dicho cálculo han sido uno para el cálculo de la matriz adjunta, otro para el cálculo del cofactor, otro para el cálculo de la inversa y, por último, otro para el cálculo del determinante 3x3.

```
G_Transpuesta := Transpuesta(G);
Inversa_G_Transpuesta := Multiplica_Matrices_20x140(Inversa,G_Transpuesta);
H := Multiplica_Matrices_20x140(Inversa_G_Transpuesta,alfaM);
```

Una vez que se cuenta con la matriz inversa, el cálculo de la matriz H se realiza mediante multiplicaciones de matrices.

Como último apartado del cálculo offline se realiza la inicialización de variables:

```
// Variables globales
gvl.C_complex := C_complex;
gvl.Matriz_P := Matriz_P;
gvl.V := Matriz_V;
gvl.Q := Q;
gvl.G := G;
gvl.H := H;


// Iniciamos las variables
gvl.u0[1] := vc_0;
gvl.u0[2] := vg_0;

gvl.u_max[1] := 6;
gvl.u_max[2] := 6;

gvl.u_min[1] := 0.1;
gvl.u_min[2] := -6;

// Equilibrio
gvl.y0[1] := alpha_0;
gvl.y0[2] := theta_0;

gvl.u_ant[1] := 0;
gvl.u_ant[2] := 0;

gvl.x_ant[1] := 0;
gvl.x_ant[2] := 0;
gvl.x_ant[3] := 0;
gvl.x_ant[4] := 0;

gvl.y_ant[1] := 0;
gvl.y_ant[2] := 0;

gvl.z_obs_ant[1] := 0;
gvl.z_obs_ant[2] := 0;

gvl.Inicializado := TRUE;
```

3.3.2. Código online en CoDeSys.

Este será el código que se ejecutará en cada periodo de muestreo seleccionado. En él se pueden diferenciar varios bloques con funciones completamente diferentes.

El primer bloque sería aquel en el que tratamos la señal analógica de entrada para convertirla desde un número entero que indica la cantidad de voltios leídos por el sensor hasta la cantidad de radianes respecto al punto de equilibrio en los que se encuentra el sistema.

Esta conversión consta de diferentes pasos, el primero de ellos será pasar el valor que detecta nuestro PLC con la configuración de 16 bits (15 bits para asignar valor y 1 bit para definir el signo) a un valor en voltios que sea más sencillo de comprender. Una vez que se cuenta con este valor en voltios, habrá que realizar una función para aproximar esos valores a los grados en los que se encuentre nuestro sistema para cada valor de tensión. Como podemos observar, esta función es diferente para cada sensor pese a que su rango de medida es el mismo (-5V a 5V), esto es debido a que en el eje de cabeceo el movimiento es mucho más reducido al tener limitaciones físicas.

Tras realizar la conversión a grados, será momento de pasar este valor a radianes y restarle los grados que nuestro proceso mide en equilibrio ya que todo el control predictivo se basa en incrementos.

Este procedimiento se ha tenido que realizar de la igual manera para el set-point, que se definirá paralelamente con otro código.

```
// Convertimos y_measure y sp

//Esto es la medida en grados 15 bits tanto del pitch como del yaw
PitchVolt := INT_TO_REAL(gvl.EntradaAI0)/32768 * 5 + 0.3;
YawVolt := INT_TO_REAL(gvl.EntradaAI1)/32768 * 5;
y_measure[1] := -81.93*PitchVolt + 218;
y_measure[2] := -71.96*YawVolt + 178.7;

// Pasamos las medidas a radianes
FOR i:=1 TO 2 DO
    y_measure[i] := (y_measure[i] - gvl.y0[i])*3.1416/180;
END_FOR

// Pasamos los objetivos pedidos en grados a radianes
sp_0[1] := (Set_Point_Pitch - gvl.y0[1])*3.1416/180;
sp_0[2] := (Set_Point_Yaw - gvl.y0[2])*3.1416/180;
```

Una vez realizados los cálculos iniciales, que nos permiten conocer el punto en el que se encuentra nuestro sistema y el punto al que debemos llevarlo, se comienza con la inicialización de variables. En primer lugar se deberá asignar el valor a la estimación de los estados, contando con que dos de ellos son mediable debido a que son el ángulo de cabeceo y el ángulo de guiñada. Para estimar los otros dos estados será necesario el uso del observador del sistema.

```
// Estimamos los estados (dalpha dtheta)
u_y_ant[1] := gvl.u_ant[1];
u_y_ant[2] := gvl.u_ant[2];
u_y_ant[3] := gvl.y_ant[1];
u_y_ant[4] := gvl.y_ant[2];

A_obs_z_obs_ant := Multiplica_Matrices_2x1(gvl.A_Obs,gvl.z_obs_ant);
B_obs_u_y_ant := Multiplica_Matrices_2x1(gvl.B_Obs, u_y_ant);

z_obs := Suma_Matrices_2x1(A_obs_z_obs_ant,B_obs_u_y_ant);

C_obs_z_obs := Multiplica_Matrices_2x1(gvl.C_Obs,z_obs);
D_obs_u_y_ant := Multiplica_Matrices_2x1(gvl.D_Obs, u_y_ant);
x_obs := Suma_Matrices_2x1(C_obs_z_obs,D_obs_u_y_ant);

// Contruimos vector de estados
x_k[1] := y_measure[1];
x_k[2] := x_obs[1];
x_k[3] := y_measure[2];
x_k[4] := x_obs[2];
```

Los siguientes pasos ya serán propios del algoritmo de control predictivo basado en modelo. El primer paso de este algoritmo será la construcción de los vectores extendidos para el set-point y el error respecto al valor estimado y el medido.

```
// Algoritmo de control
C_TRMS_x_k := Multiplica_Matrices_2x1(gvl.C_TRMS, x_k);
bias := Resta_Matrices_2x1(y_measure,C_TRMS_x_k);

i:=1;
FOR i := 1 TO 2*gvl.p DO
    IF i MOD 2 = 1 THEN
        bias_ext[i] := bias[1]; // Asignar el valor de la columna 1 de Aux si j es impar
        sp[i] := sp_0[1];
    ELSE
        bias_ext[i] := bias[2]; // Asignar el valor de la columna 2 de Aux si j es par
        sp[i] := sp_0[2];
    END_IF
END_FOR
```

Seguido a la construcción de estos vectores, el algoritmo sigue realizando operaciones matriciales en las que son necesarias el uso de las funciones diseñadas para permitir el cálculo tanto de la multiplicación como suma y resta de matrices.

```
A_TRMS_x_ant := Multiplica_Matrices_4x1(gvl.A_TRMS,gvl.x_ant);
B_TRMS_u_ant := Multiplica_Matrices_4x1(gvl.B_TRMS,gvl.u_ant);
A_TRMS_x_ant_B_TRMS_u_ant := Suma_Matrices_4x1(A_TRMS_x_ant,B_TRMS_u_ant);
v := Resta_Matrices_4x1(x_k,A_TRMS_x_ant_B_TRMS_u_ant);

P_x_k := Multiplica_Matrices_140x1(gvl.Matriz_P,x_k);
C_Matriz_V := Multiplica_Matrices_140x4(gvl.C_complex,gvl.V);
C_Matriz_V_v := Multiplica_Matrices_140x1(C_Matriz_V,v);
bias_ext_C_Matriz_v_v := Suma_Matrices_140x1(bias_ext,C_Matriz_V_v);
Q_u_ant := Multiplica_Matrices_140x1(gvl.Q,gvl.u_ant);
y_free_1 := Suma_Matrices_140x1(P_x_k,bias_ext_C_Matriz_v_v);
y_free_2 := Suma_Matrices_140x1(y_free_1,Q_u_ant);
sp_y_free := Resta_Matrices_140x1(sp,y_free_2);
du_next := Multiplica_Matrices_20x1(gvl.H, sp_y_free);

// Aplicamos el primer incremento de acción de control previsto
du[1] := du_next[1];
du[2] := du_next[2];
```

Tras llegar a calcular la serie de incrementos de la acción de control a tomar para todo el horizonte de predicción, debemos quedarnos con la primera predicción y sumarle la acción de control anterior. Por seguridad se ha añadido un bloque de saturación para evitar el envío de tensiones superiores a los límites. Una vez que ya hemos obtenido las tensiones a aplicar en cada uno de los motores del sistema, deberemos convertir este número real a un entero de 15 bits para, así, poder enviar esta tensión desde el PLC.

```
// Calculamos acciones de control finales
U_ant_du := Suma_Matrices_2x1(gvl.u_ant,du);
u := Suma_Matrices_2x1(U_ant_du,gvl.u0);

// Saturamos las salidas
i:=1;
FOR i := 1 TO 2 DO
    IF u[i] > gvl.u_max[i] THEN
        u[i] := gvl.u_max[i];
    ELSIF u[i] < gvl.u_min[i] THEN
        u[i] := gvl.u_min[i];
    END_IF
END_FOR

// Escribimos la tensión a aplicar -> Añadir cambios de V a numerito
gvl.SalidaAO0 := LREAL_TO_INT(u[1]*32768/10);;//REAL_TO_INT((u[1]-0.3)*32768/10);
gvl.SalidaAO1 := LREAL_TO_INT(u[2]*32768/10);
```

Por último, este algoritmo necesita la actualización de valores para la iteración posterior.

Al igual que en el control mediante Matlab, para el control en CoDesys se ha implementado un código que permite la creación de una trayectoria a seguir mediante la definición de setpoints que cambian a lo largo del tiempo. Ya que CoDeSys no trabaja con vectores temporales como Matlab sino que ejecuta un código cada X periodo de tiempo, ha sido necesario el siguiente código que se ejecuta cada 1 segundo. Este código cuenta con una variable llamada "T_espera" que tiene un valor de 15 segundos debido a que para mostrar el funcionamiento del sistema, no podrá existir una persona que lo mantenga en el punto de equilibrio al comenzar con el control, por lo tanto, este tiempo permite al sistema llegar a ese punto de equilibrio de forma autónoma.

```
IF (gvl.START AND gvl.I5 AND NOT gvl.EMERGENCIA_HMI)= TRUE THEN
    IF T = 0 THEN
        gvl.Ref_Pitch := 0;
        gvl.Ref_Yaw := 0;
    ELSIF T >= 10 + gvl.T_espera AND T < 20 + gvl.T_espera THEN
        gvl.Comienza := TRUE;
        gvl.Ref_Pitch := 0;
        gvl.Ref_Yaw := 30;
    ELSIF T >= 20 + gvl.T_espera AND T < 30 + gvl.T_espera  THEN
        gvl.Ref_Pitch := 30;
        gvl.Ref_Yaw := 30;
    ELSIF T >= 30 + gvl.T_espera AND T < 40 + gvl.T_espera THEN
        gvl.Ref_Pitch := 30;
        gvl.Ref_Yaw := 70;
    ELSIF T >= 40 + gvl.T_espera AND T < 50 + gvl.T_espera THEN
        gvl.Ref_Pitch := 15;
        gvl.Ref_Yaw := 70;
    ELSIF T >= 50 + gvl.T_espera AND T < 60 + gvl.T_espera THEN
        gvl.Ref_Pitch := 15;
        gvl.Ref_Yaw := -100;
    ELSIF T >= 60 + gvl.T_espera AND T < 70 + gvl.T_espera THEN
        gvl.Ref_Pitch := -15;
        gvl.Ref_Yaw := -100;
    ELSIF T >= 70 + gvl.T_espera AND T < 80 + gvl.T_espera THEN
        gvl.Ref_Pitch := -15;
        gvl.Ref_Yaw := 0;
    ELSIF T >= 80 + gvl.T_espera AND T < 90 + gvl.T_espera THEN
        gvl.Ref_Pitch := 30;
        gvl.Ref_Yaw := 0;
    ELSIF T >= 90 + gvl.T_espera AND T < 100 + gvl.T_espera THEN
        gvl.Ref_Pitch := 30;
        gvl.Ref_Yaw := 115;
    ELSIF T >= 100 + gvl.T_espera AND T < 110 + gvl.T_espera THEN
        gvl.Ref_Pitch := 0;
        gvl.Ref_Yaw := 115;
    ELSIF T >= 110 + gvl.T_espera THEN
        gvl.Ref_Pitch := 0;
        gvl.Ref_Yaw := 0;
    END_IF

    T := T + 1;
END_IF
```

3.3.3. Configuración de tareas en CoDeSys

Una vez desarrollado el código a ejecutar, se ha llegado a la conclusión de que para el control de nuestro helicóptero necesitaremos ejecutar una vez el código offline y, luego, en cada instante de tiempo se deberá ejecutar el control predictivo basado en modelo.

Para ello ha sido necesaria la implementación de un POU en contactos. Gracias a este POU y la definición de una variable booleana que nos indica si el sistema ha sido inicializado, podremos ejecutar una única vez el código offline y, posteriormente, el código referente al control. Como podemos observar, el bloque del control tiene otras 2 entradas las cuales contienen el valor del set-point que el sistema debe alcanzar y se definen dentro de script de creación de referencias.

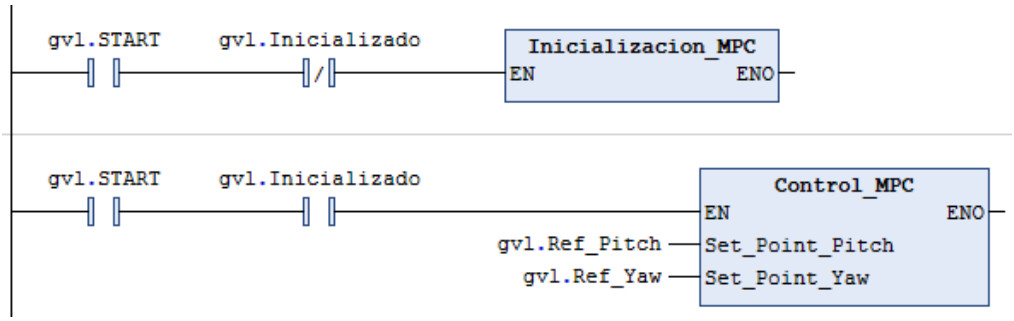

Figura 32. Código Ladder para el control MPC.

Ya que el control debe tener una periodicidad de 50 milisegundos y este periodo sería completamente contraproducente para la generación de referencias, se ha optado por la creación de 2 tareas diferentes. Una de ellas, llamada “Control”, es la que se encarga de ejecutar el código Ladder de la **Figura 32** y otro de ellas, llamada “Referencias”, es la que ejecuta el código para la creación de referencias y su periodo de ejecución es cada 1 segundo.

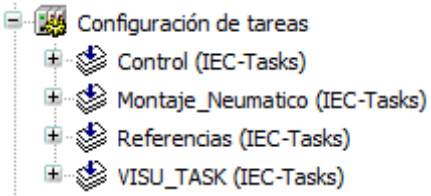

Figura 33. Tareas CoDeSys

Capítulo 4

Automatización del montaje neumático

En este capítulo se realizará una explicación paso a paso de la dinámica del montaje neumático utilizado para la reproducción del proceso. Más adelante, se procederá a explicar los requisitos que debe cumplir la solución alcanzada.

4.1. Descripción del montaje neumático.

Este será el proceso a automatizar para demostrar la capacidad que posee un PLC actual para realizar tanto el control de un sistema multivariable complejo como la automatización paralela de un sistema como este.

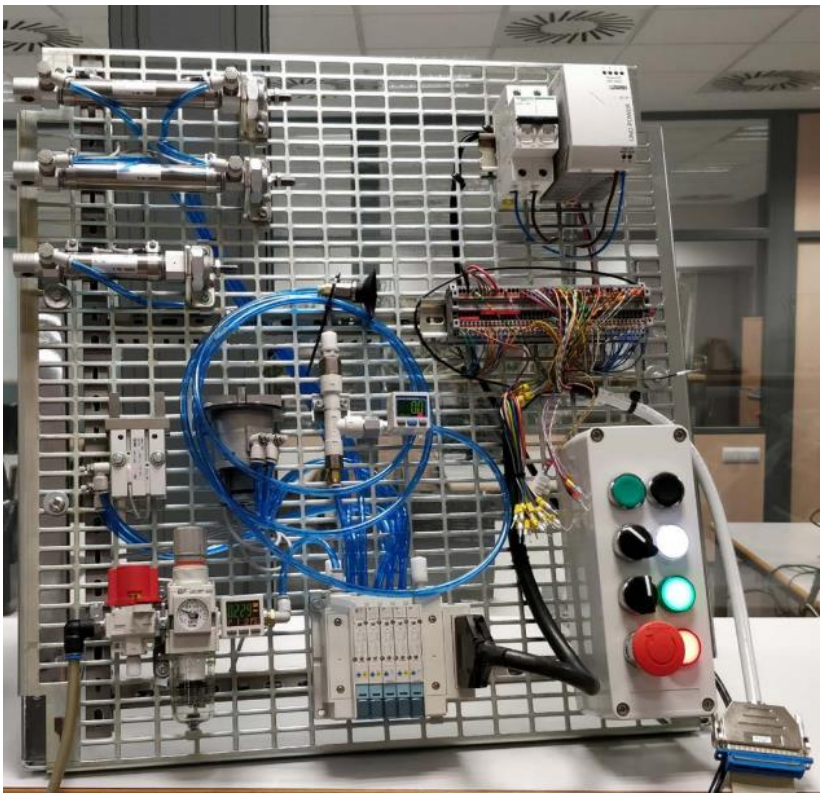

Figura 34. Montaje Neumático

Este montaje cuenta con diferentes bloques fácilmente diferenciables. Estos bloques son:

- Puesto de control.

Este puesto de control está definido por una serie de botoneras y luces que permiten tanto la introducción manual de señales en el sistema como la visualización de estados del proceso en tiempo real. La información que aporte cada uno de los leds y las acciones que ejecuten cada uno de los botones será definido dependiendo del fin de la automatización realizada.

Figura 35. Puesto de control

Este subsistema cuenta con 6 entradas digitales y con 3 salidas digitales.

Tabla 3. Entradas y Salidas del puesto de control

Entradas	Tipo	Dirección	Descripción
I0	Bool	%IX0.0	Botonera pulsador verde
I1	Bool	%IX0.1	Botonera pulsador negro
I2	Bool	%IX0.2	Botonera selector 2 posiciones (Derecha)
I3	Bool	%IX0.3	Botonera selector 3 posiciones I (Izquierda)
I4	Bool	%IX0.4	Botonera selector 3 posiciones II (Derecha)
I5	Bool	%IX0.5	Botonera seta (N.C.)

Salidas	Tipo	Dirección	Descripción
Q8	Bool	%QX1.0	Botonera piloto verde
Q9	Bool	%QX1.1	Botonera piloto blanco
Q10	Bool	%QX1.2	Botonera piloto rojo

- Cilindros neumáticos.

Este subsistema cuenta con 3 cilindros neumáticos en los que dos de ellos son de simple efecto (simplemente se controla a la hora de sacarlos) y uno de ellos es de doble efecto (se controla tanto a la hora de sacar como meter).

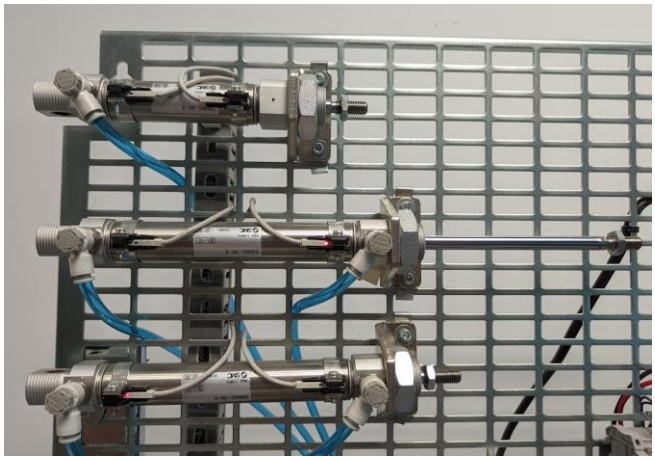

Figura 36. Cilindros neumáticos

Este subsistema cuenta con 4 entradas digitales y con 4 salidas digitales.

Tabla 4. Entradas y Salidas de los cilindros neumáticos

Entradas	Tipo	Dirección	Descripción
I6	Bool	%IX0.6	Cilindro superior sensor fuera
I7	Bool	%IX0.7	Cilindro centro sensor fuera
I8	Bool	%IX1.0	Cilindro centro sensor dentro
I9	Bool	%IX1.1	Cilindro inferior sensor fuera

Salidas	Tipo	Dirección	Descripción
Q0	Bool	%QX0.0	Sacar cilindro superior
Q1	Bool	%QX0.1	Sacar cilindro centro
Q2	Bool	%QX0.2	Introducir cilindro centro
Q3	Bool	%QX0.3	Sacar cilindro inferior

- Pinza, cilindro rotativo y ventosa.

Este subsistema cuenta con una mayor variedad de elementos al contar tanto con una pinza neumática, con un cilindro rotativo y con una ventosa que permite coger objetos.

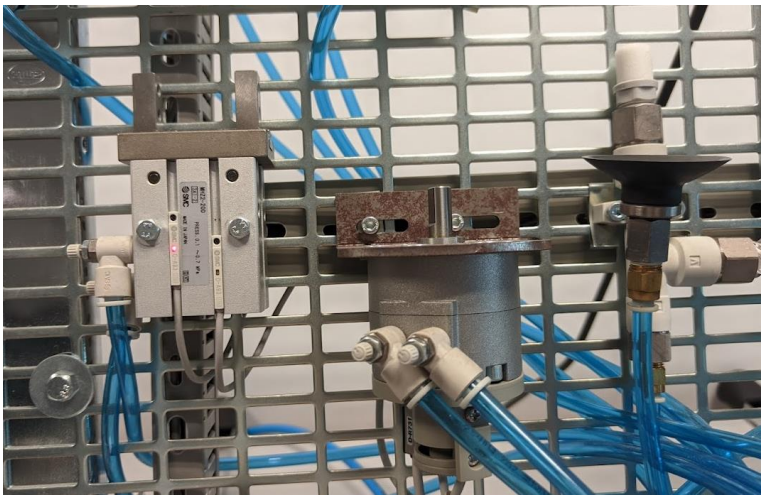

Figura 37. Pinza, cilindro rotativo y ventosa

Este subsistema cuenta con 6 entradas digitales y con 4 salidas digitales.

Tabla 5. Entradas y Salidas de la pinza, el cilindro rotativo y la ventosa

Entradas	Tipo	Dirección	Descripción
I10	Bool	%IX1.2	Pinza abierta
I11	Bool	%IX1.3	Pinza cerrada
I12	Bool	%IX1.4	Cilindro rotativo inicio
I13	Bool	%IX1.5	Cilindro rotativo final
I14	Bool	%IX1.6	Succión de aire (N.C.)
I15	Bool	%IX1.7	Objeto en ventosa (N.C.)

Salidas	Tipo	Dirección	Descripción
Q4	Bool	%QX0.4	Cerrar pinza
Q5	Bool	%QX0.5	Abrir pinza
Q6	Bool	%QX0.6	Cilindro rotativo (horario)
Q7	Bool	%QX0.7	Vacío en ventosa

4.2. Automatización

Para la realización de la optimización del montaje neumático se ha optado por seguir la estructura que marca la guía GEMMA, que es la guía de estudio de modos de marchas y paradas. Esta guía contempla una gran cantidad de bloques en los que se pueden encapsular las funciones que se realizan o son viables durante el funcionamiento de un proceso automatizado.

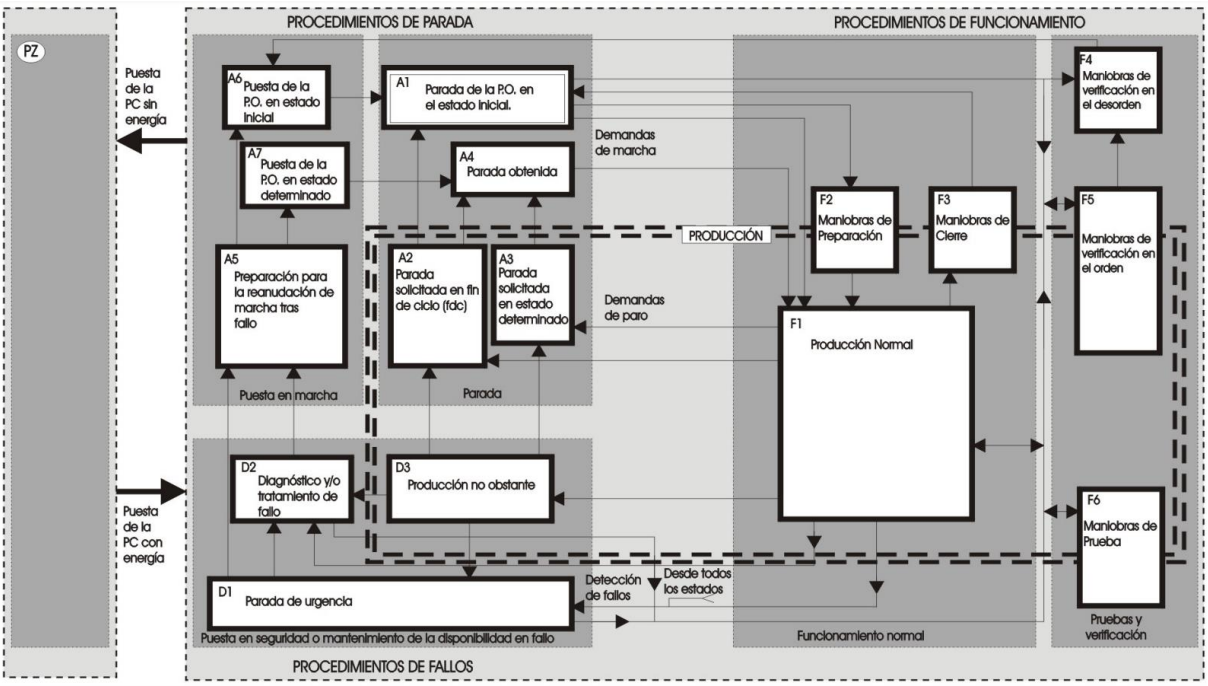

Figura 38. Diagrama de la guía GEMMA. Fuente: [6]

En nuestro caso se han seleccionado los procedimientos que más se ajustaban con nuestro proceso, estos procedimientos son de inicio a fin:

- F2 o maniobras de preparación. En este bloque se llevan a cabo las operaciones necesarias antes de llegar al funcionamiento a plena capacidad.
- F1 o producción normal. En este bloque se contienen las operaciones que realiza cíclicamente el proceso durante su funcionamiento a plena capacidad.
- D1 o parada de emergencia. Es el estado en el que debe entrar el sistema de parada inmediata debido a un fallo, suele ejecutarse al detectarse el pulsado de una seta de emergencia.

Para su diseño se ha utilizado el modelo de representación gráfica GRAFCET, en su diseño ha sido necesaria la incorporación de encapsulados en los que se desarrollen los procedimientos anteriormente descritos. El GRAFCET diseñado luce de la siguiente forma:

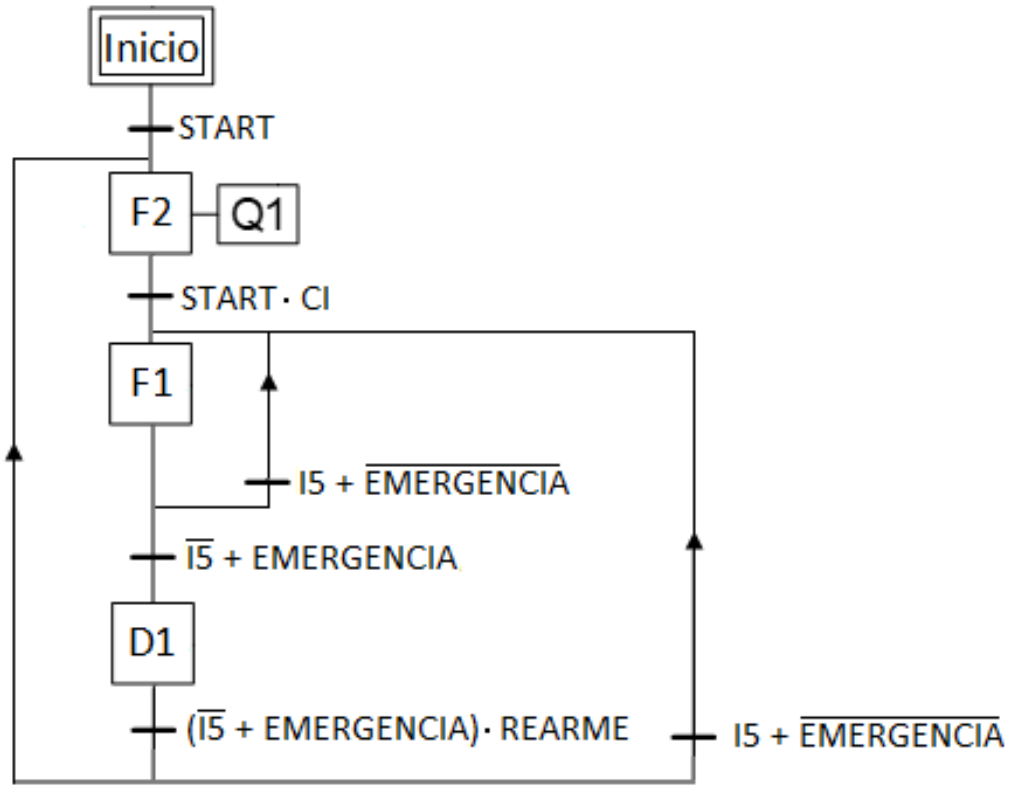

Figura 39. GRAFCET diseñado para GEMMA

Como se puede observar en la **Figura 40,** para la implementación del GRAFCET diseñado, ha sido necesaria la utilización del tipo de código llamado SFC (Sequential Function Chart) que se asemeja al propio GRAFCET.

Nuestro sistema en primer lugar cuenta con un estado en el que se mantiene en reposo hasta que se detecte la activación del botón de encendido, este botón podrá pulsarse tanto físicamente como en el HMI implementado (se explica en el apartado **4.2.4. Interfaz humano-máquina**). Una vez que se haya detectado la activación del sistema, se pasará al subsistema F2 donde se realizarán todas las operaciones necesarias para comprobar que el sistema se encuentra en su condición inicial y, así, poder seguir con su producción normal. En el caso de que durante este proceso se pulse la seta de emergencia, se pasará directamente al estado D1 donde se desactivan todas las señales y se esperará al rearme o desactivación del sistema.

Si se finaliza el proceso F2 sin imprevistos, se comenzará con el macro estado F1 donde se realizan las operaciones de producción normal. Este macro estado es cíclico siempre y cuando no se active la seta de emergencia.

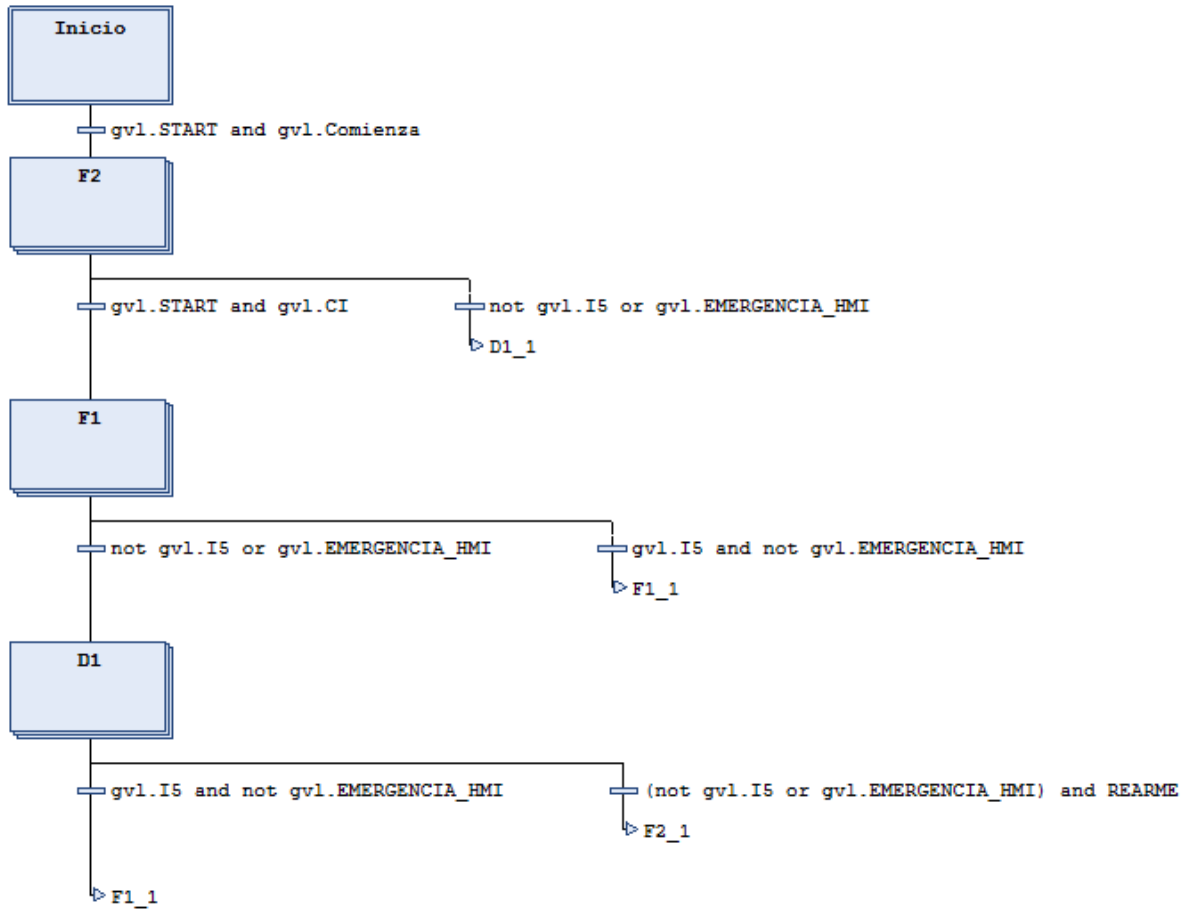

Figura 40. SFC de la guía GEMMA

4.2.1. F2 o maniobras de preparación

El macro estado F2 es aquel en el que se realizan todas las operaciones necesarias para comprobar que el sistema se encuentra en sus condiciones iniciales, para así, poder pasar al funcionamiento normal sin problemas. Este macro estado se ha diseñado utilizando el modelo de representación gráfica llamado GRAFCET. El GRAFCET resultante de este proceso es el siguiente.

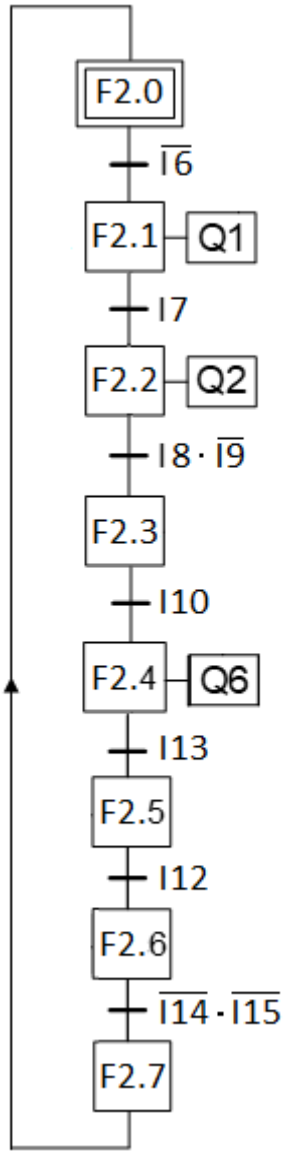

Figura 41. GRAFTCET diseñado para F2

El macro estado implementado consiste en la comprobación de que los cilindros de simple efecto se encuentran en el interior y, para el caso del cilindro de doble efecto, en primer lugar se saca y se comprueba que llega al final de carrera para después introducirlo y comprobar que también llega al final de carrera interior.

Una vez que se ha comprobado el correcto funcionamiento de los cilindros neumáticos, se pasa a comprobar la apertura de la pinza neumática. Posteriormente se comprueba que la mesa giratoria funcione correctamente en ambos sentidos y, por último, se comprueba que la ventosa no esté activa ni tenga ningún elemento adherido a ella.

En el software CoDeSys este funcionamiento se ha implementado mediante un SFC debido a que es la forma más similar al diseño realizado y tiene la siguiente forma.

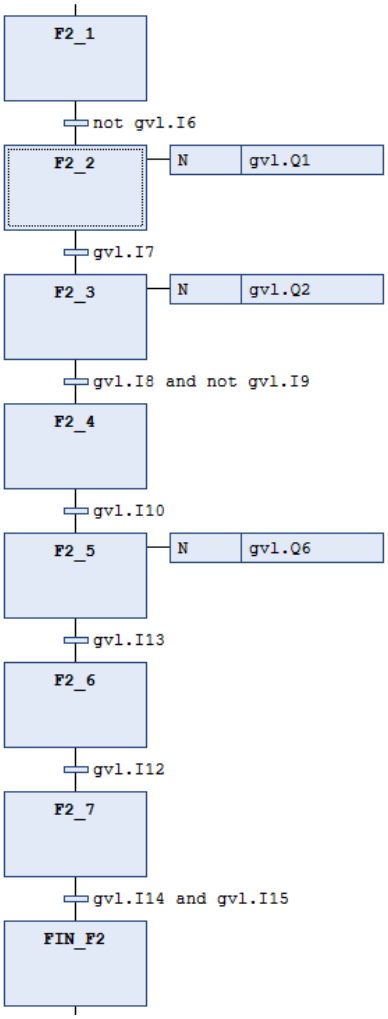

Figura 42. SFC del Macrosistema F2

4.2.2. F1 o producción normal

Una vez más, este macro estado se ha diseñado utilizando el modelo de representación gráfica llamado GRAFCET. El funcionamiento de esta etapa es la siguiente:

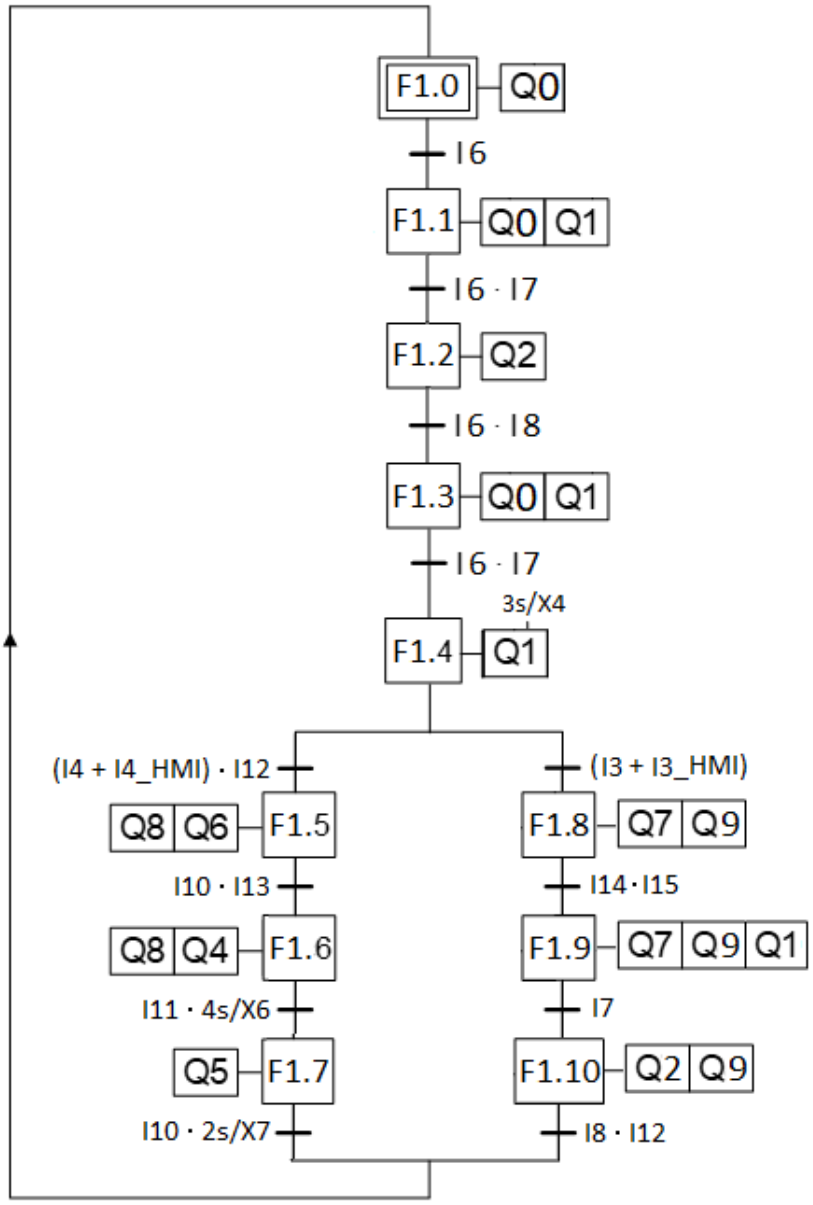

Figura 43. GRAFCET diseñado para F1

La producción normal comienza con una secuencia que se encarga de llevar el pistón superior a su final de carrera exterior para luego, mantenerlo en el exterior mientras también se saca el cilindro central. Una vez este se encuentre en el exterior, se permite que ambos cilindros vuelvan a su punto interior actuando únicamente sobre el cilindro central al ser de doble efecto.

La secuencia sigue sacando una vez más ambos cilindros para, posteriormente, dejar que el cilindro superior vuelva al interior mientras que el cilindro central se mantiene durante 3 segundos en su sensor final de carrera externo.

Tras esta secuencia, el operario debe haber seleccionado que tipo de producto se debe realizar ya que dependiendo de la posición de un selector (tanto físico como en el HMI), se realizará una secuencia distinta dependiendo de la selección realizada.

En una de las opciones, las operaciones que se realizan son la de girar la mesa para cerrar la pinza, mantenerla cerrada durante 4 segundos para posteriormente abrirla. Todo este proceso se realizará con el piloto de luz verde activo.

En el caso de que la seleccionada sea la otra opción, las operaciones realizadas serán la de activar la ventosa, una vez que esté activa y mantenga un objeto, se sacará el cilindro central hasta su final de carrera exterior para posteriormente introducirlo. Todo este proceso se realizará con el piloto de color blanco activo.

En el software CoDeSys este funcionamiento se ha implementado mediante un SFC debido a que es la forma más similar al diseño realizado y tiene la siguiente forma.

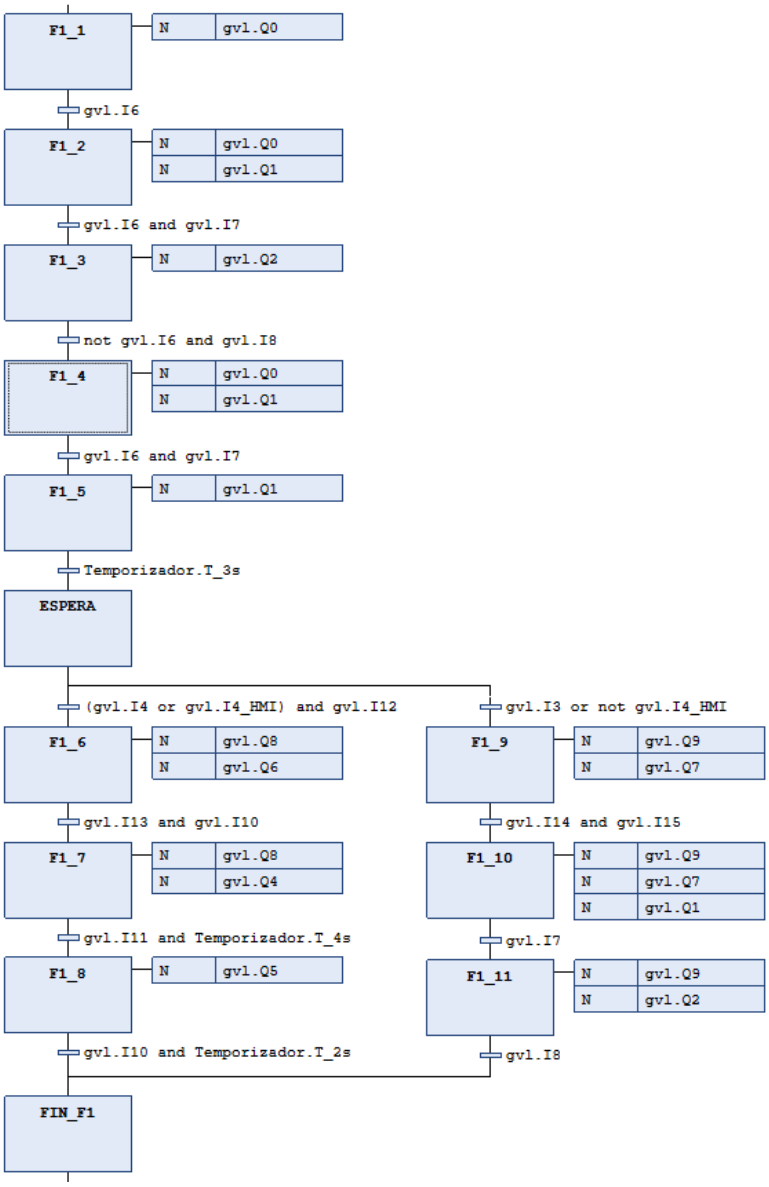

Figura 44. SFC del Macrosistema F1

4.2.3. Temporizadores y variables internas.

El uso de temporizadores para el correcto funcionamiento del diseño se ha realizado mediante el uso de bloques "TON" los cuales realizan la función de temporizadores con retardo a la conexión. El uso de estos bloques ha permitido el paso de estado una vez que cierto periodo de tiempo había transcurrido desde su activación. Para su implementación ha sido necesaria la creación de un POU en lenguaje Ladder (LD) o contactos.

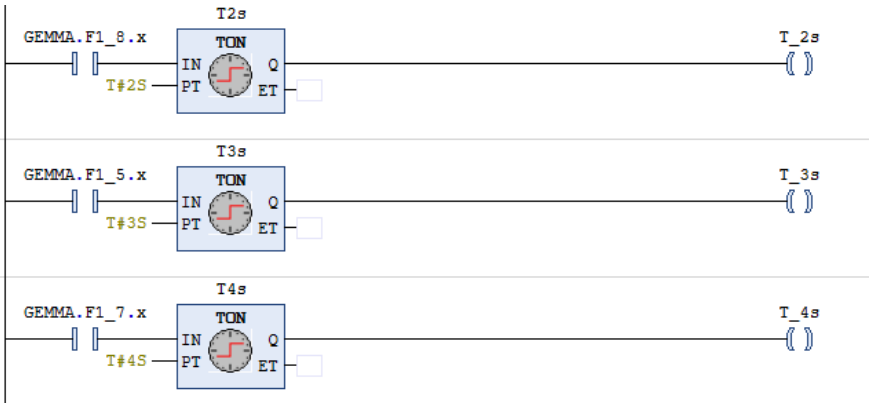

Figura 45. Bloques TON para la implementación de temporizadores

Ya que se contaba con la existencia de un POU en contactos, también se ha utilizado para la declaración de variables complejas. Una de estas variables sería la que nos indica si el sistema se encuentra en las condiciones iniciales. Para la definición de esta variable se ha dividido el montaje neumático en 3 bloques.

- CI_Cilindros. Se encarga de comprobar que los 3 cilindros se encuentren activando el sensor de su interior.
- CI_Mesa. Transmite la información de que la mesa se encuentra en su posición de inicio del giro, se añade una comprobación de que el sensor de final de giro no está activo.
- CI_Ventosa. Esta variable comprueba que la ventosa no tenga la succión activa y, por lo tanto, no tenga ningún elemento adherido.

Para la activación de la variable global de condiciones iniciales, las 3 variables anteriores deben estar activas y, además, el proceso de deberá encontrar en el fin del macro estado F2.

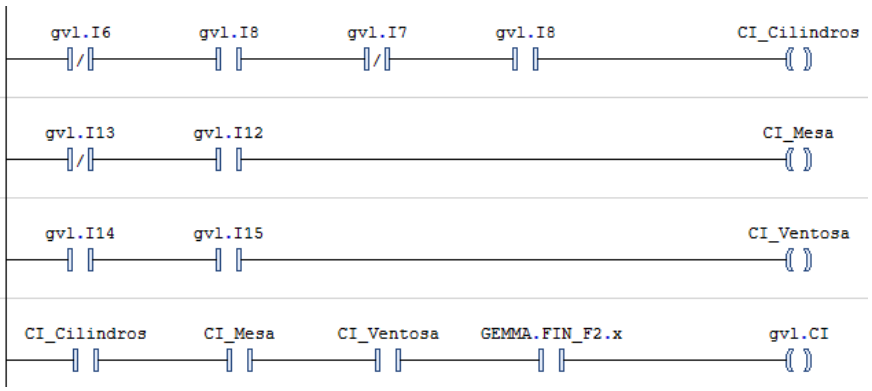

Figura 46. Ladder de la CI

También se ha requerido de la creación de una variable global de inicio de proceso. Para la activación de esta, el operario deberá pulsar el botón físico de inicio o el botón táctil que se encuentra en el HMI. Además, para la activación del inicio, no podrá estar activo el estado de emergencia y, por lo tanto, no podrán estar activos ni la seta de emergencia física ni el botón del HMI definido para este uso.

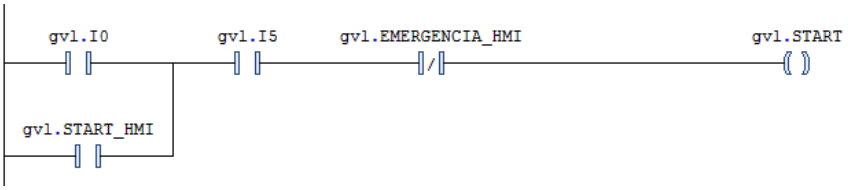

Figura 47. Ladder de la variable START

4.2.4. Interfaz humano-máquina (HMI)

En este apartado se presentará la interfaz humano-máquina propuesta para el control de todo el proceso implementado. Esta interfaz se ha dividido en dos partes claramente diferenciadas. Estas partes serían por un lado la parte superior, referente al control avanzado de la maqueta del helicóptero y, por otro lado en la parte inferior, contamos con el bloque propio de la automatización del montaje neumático.

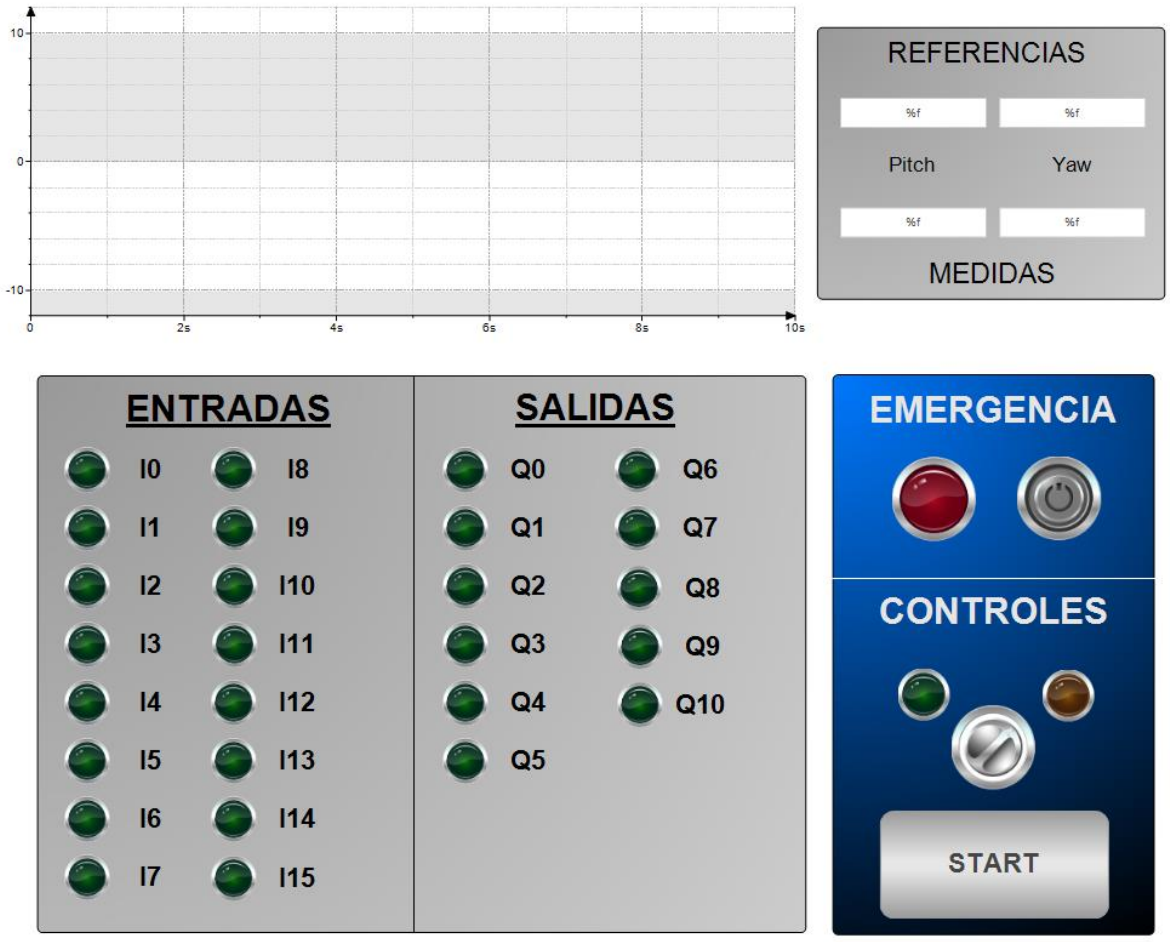

Figura 48. Interfaz visual implementada

En el bloque superior, con información sobre el control avanzado, podemos observar la traza del movimiento que esté ejecutando nuestro sistema en ese momento. Además, también se puede localizar un bloque en el que se encuentran tanto los valores numéricos de la referencia y las medidas para, así, poder comprobar que la traza es real.

El bloque inferior consta de otras dos partes. Una de ellas sería el puesto de mando donde se encuentra el botón de inicio que permite que tanto el helicóptero comience con su trayectoria como que el montaje neumático comience con su proceso.

Además, en este puesto de control se encuentra un selector necesario para la selección de una alternativa a la hora de automatizar u otra. También cuenta con el botón de emergencia, el cual parará cualquier proceso que se esté llevando a cabo.

El otro sub-bloque que conforma el bloque inferior sería el visualizador de entradas y salidas digitales activas. Se ha optado por un simple visualizador debido a que un operario no puede estar pendiente de muchos movimientos en caso de que todo vaya bien, por lo tanto, se ha optado por unas luces que nos aportan información de qué parte del proceso se está llevando a cabo.

Una vez que la interfaz estaba totalmente desarrollada, se ha optado por la habilitación de la visualización web de la misma. Para configurar dicha visualización ha sido necesario seleccionar como visualización de arranque el elemento en el que se encuentre nuestro HMI y, además, se ha seleccionado el escalado isótropo al ser aquel que se adapta al navegador y mantiene la proporción de la visualización sin ser afectado por el tamaño de la pantalla en el que se acceda a dicha visualización.

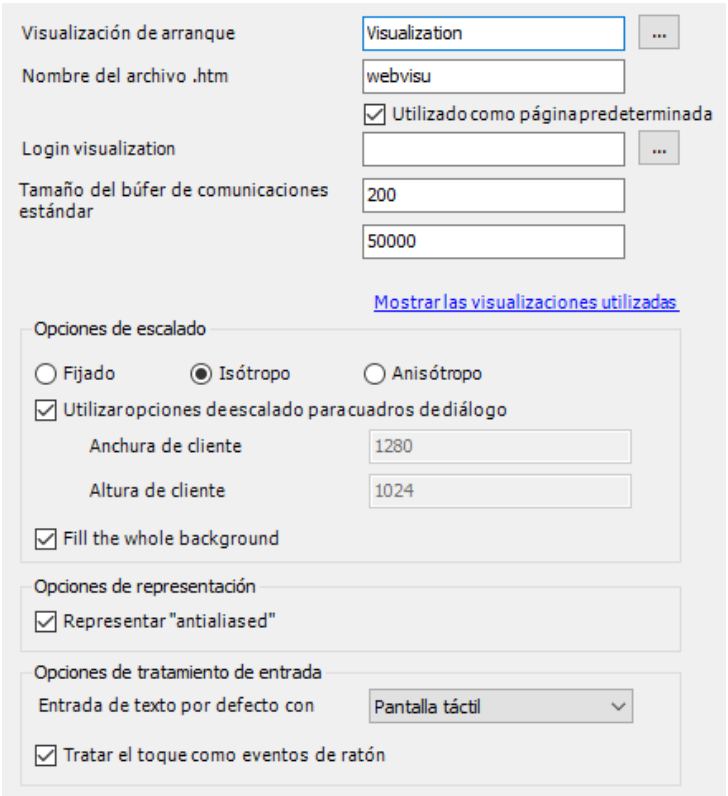

Figura 49. Configuración visualizador Web

Para acceder desde cualquier dispositivo a la visualización web basta con escribir en cualquier navegador la siguiente dirección “172.16.191.75:8080/webvisu.htm”. Esta dirección está formada por tres elementos diferentes, el primero de ellos sería la dirección IP del PLC seguido del puerto utilizado y, para finalizar, se debe seleccionar el nombre del archivo de visualización web.

Capítulo 5

Conclusiones

El presente proyecto se ha ejecutado con éxito. Se han logrado todos los objetivos fijados en un primer momento y se ha logrado una metodología a seguir para la implantación de controles avanzados en PLCs industriales.

Durante el desarrollo de las funciones necesarias para llevar a cabo el control predictivo basado en modelo siempre se ha tenido en cuenta que uno de nuestros objetivos era que el control fuese fácilmente adaptable a cualquier tipo de proceso. Por lo tanto, todas las funciones se han programado de forma que únicamente será necesaria la modificación de las dimensiones de las matrices con las que se trabaja o, como alternativa, se podrá eliminar la barrera de seguridad que en este proyecto se ha mantenido e impedía el envío de matrices de dimensiones variables como parámetro.

A su vez, se ha implementado correctamente una automatización de elementos industriales que se realizaba paralelamente al control avanzado. También se ha propuesto una interfaz humano-máquina la cual cuenta con todo lo necesario para el control y supervisión del proceso.

Para finalizar, en este trabajo fin de máster se han establecido unas bases sólidas para futuros proyectos de implementación de controles avanzados en PLCs industriales utilizando CoDesys, ya que se ha resuelto la mayor problemática con la que cuenta este software.

BIBLIOGRAFÍA

[1] International Electrotechnical Commission et al., Grafcet specification language for sequential function charts, International Standard, IEC 60848 (2002), 94.

[2] Ikonen, E. (2017). Model Predictive Control and State Estimation.

[3] Roig Roig, J. V. (2022). Apuntes de la asignatura Ingeniería de Control del Máster Universitario en Ingeniería Industrial. Universitat Politècnica de València.

[4] Sanchís Sáez, J., & García-Nieto Rodríguez, S. (2022). Apuntes de la asignatura Control Industrial Avanzado del Máster Universitario en Ingeniería Industrial. Universitat Politècnica de València.

[5] Wang, L. (2009). Model Predictive Control System Design and Implementation Using MATLAB® [electronic resource] (1st ed. 2009.). Springer London. *https://doi.org/10.1007/978-1-84882-331-0*

[6] Emilio García Moreno. (2002). Automatización de procesos industriales. Universitat Politècnica de València.

[7] Camacho, E.F., y C. Bordóns. (2004). "Model Predictive Control". Springer.

[8] Es.mathworks.com. (2023). Simulation and Structured Text Generation Using PLC Coder. MATLAB & Simulink- MathWorks España. [online] Disponible en:

https://es.mathworks.com/help/mpc/ug/simulation-and-structured-text-generation-using-plc-coder.html

[9] 3S-Smart Software Solutions GmbH CODESYS. (2023). https://www.codesys.com

[10] PLCnext Technology Community. (2023). https://www.plcnext-community.net/

[11] ResearchGate. (2023). https://www.researchgate.net/figure/Block-diagram-for-model-predictive-control-1_fig4_255402709

[12] Universidad Nacional de Colombia. (2023) https://minas.medellin.unal.edu.co/noticias/2682-adquisicion-de-matlab-y-simulink

[13] Wikipedia Commons. (2023). https://commons.wikimedia.org/wiki/File:Codesys_Logo.svg

DESARROLLO E IMPLEMENTACIÓN DE ALGORITMOS AVANZADOS DE CONTROL MEDIANTE PLCS INDUSTRIALES.
APLICACIÓN A LA AUTOMATIZACIÓN Y CONTROL HÍBRIDO DE UN PROCESO INDUSTRIAL MEDIANTE UN AUTÓMATA PLCNEXT TECHNOLOGY.

DOCUMENTO Nº2: PRESUPUESTO

1.1. Introducción

El presente documento tiene como objetivo la descripción detallada de todos los elementos que se han requerido para la realización de este trabajo fin de máster.

Con el objetivo de que el coste del proyecto se represente lo más detallado posible, se ha dividido en tres apartados principales.

- **Elementos hardware**. Este apartado está formado por todos los elementos físicos necesarios para la realización del proyecto anteriormente expuesto.
- **Elementos software**. Este apartado se ha formado para crear una unidad únicamente comprendida por los diferentes programas utilizados durante el desarrollo del presente trabajo fin de máster.
- **Mano de obra**.

Estos apartados se desarrollarán a continuación para, una vez descritos, poder presentar el presupuesto final.

1.2. Desarrollo del presupuesto

Elementos hardware

Tabla 6. Presupuesto elementos hardware

Código	Unidades	Descripción del material empleado	Precio de compra (€)	Rendimiento	€/unidad	Importe (€)
MA01	h	PLCnext de Phoenix Contact + módulos	820.00	200	0.02	3.80
MA02	h	Maqueta de helicóptero (Twin Rotor)	1200.00	480	0.03	13.33
MA03	h	Bancada Neumática	905	480	0.02	10.06
MA04	h	Ordenador de sobremesa con periféricos	700	200	0.02	3.24
MA05	h	Ordenador portátil	1300	280	0.03	8.43
		Total				38.85 €

Para los elementos mostrados en la tabla superior al ser elementos con precios de adquisición, se ha estimado un tiempo de vida útil de 5 años, para así, únicamente cargar sobre el cliente el porcentaje de uso asignado a este proyecto.

$$€/unidad = \frac{Precio\ de\ adquisición}{24 \cdot 30 \cdot 12 \cdot 5}$$

Los precios mostrados en la tabla superior hacen referencia a los precios por los que la universidad adquirió los diferentes elementos. Si quisiéramos replicar el proyecto a precios actuales, estos serían más elevados.

Algún ejemplo sobre el que se basa la afirmación anterior es el hecho de que actualmente el controlador lógico PLCnext de Phoenix Contact sin ningún tipo de módulo se encuentra por 673.67€ (IVA incluido) en la página web del distribuidor RS (https://es.rs-online.com/web/p/controladores-plcs-y-automatas/2055853).

Por otro lado, tanto la maqueta de helicóptero como la bancada neumática son elementos definidos expresamente para la utilización en la docencia impartida en la universidad. En el caso de la bancada neumática también es importante recalcar que es necesaria una instalación de aire a presión o, como mínimo, un compresor con el que actuar sobre la misma.

Elementos software

Tabla 7. Presupuesto elementos software

Código	Unidades	Descripción del material empleado	Rendimiento	€/unidad	Importe (€)
MA06	h	CODESYS V3.5 SP18 Patch 4	200	Gratuita	0.00
MA07	n	CODESYS Control for PLCnext MC SL	1	55.00	55.00
MA08	h	MATLAB	250	0.10	25.00
MA09	h	Simulink	30	0.15	4.50
MA10	h	Adobe Photoshop CC	5	0.03	0.15
		Total			84.65 €

Lo precios de la tabla superior se han obtenido de las siguientes fuentes:

- En primer lugar, el módulo que hace compatible CODESYS con el PLC seleccionado es un elemento que podemos obtener de forma gratuita si creamos una cuenta en la web, pero para evitar fallos se añade su precio normal. Este precio es unitario ya que se debe adquirir para poder trabajar con nuestro PLC.
- Sitio web oficial de Adobe para la obtención del precio de la licencia de Photoshop. Con un precio para empresas de 24.19€/mes, el precio horario para este proyecto sería de:
  $24.19 \div (30 \cdot 24) = 0.034\ €$
- Sitio web oficial de MathWorks para la obtención del precio de la licencia tanto de Matlab como de Simulink. Con un precio de licencia standard anual de Matlab de 860€ y un precio de licencia standard anual de Simulink de 1300€, el precio horario para este proyecto sería de:

  - Matlab: $860 \div (30 \cdot 24 \cdot 12) = 0.10\ €$

  - Simulink: $1300 \div (30 \cdot 24 \cdot 12) = 0.15\ €$

Mano de obra

Tabla 8. Presupuesto de la mano de obra

Código	Unidades	Descripción del material empleado	Rendimiento	€/unidad	Importe (€)
MO01	h	Titulado con Máster en Ingeniería Industrial especialidad en Control, Automatización y Robótica	486	20.00	9720.00
MO02	h	Técnico de laboratorio	10	25.00	250.00
		Total			9,970.00 €

1.3. Presupuesto final

Una vez desarrollados los elementos necesarios para la realización del proyecto, se cuentan con los datos necesarios para la creación del presupuesto final.

Tabla 9. Presupuesto final

Código	Unidades	Descripción del material empleado	Rendimiento	€/unidad	Importe (€)
UD01	n	Elementos hardware	1	38.85	38.85
UD02	n	Elementos software	1	84.65	84.65
UD03	h	Mano de obra	1	9970.00	9970.00

Total Presupuesto de Ejecución Material	10,093.50 €
Gastos Generales (15%)	1,514.03 €
Beneficio Industrial (6%)	605.61 €
Total Presupuesto Base de Licitación	12,213.14 €

El Presupuesto Base de Licitación asciende a la cantidad de: DOCE MIL DOSCIENTOS TRECE EUROS CON CATORCE CÉNTIMOS.