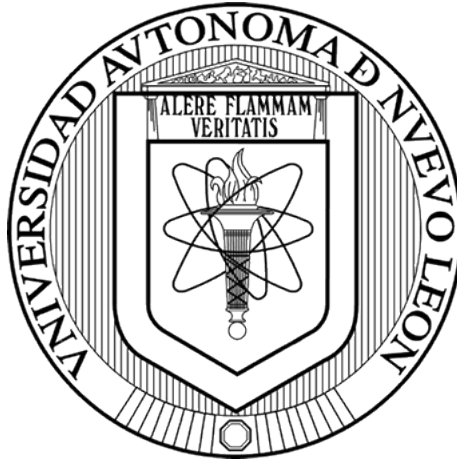


UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN
FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA
DIVISIÓN DE ESTUDIOS DE POSGRADO



INTEGRACIÓN DE UN PROCESADOR DIGITAL DE SEÑALES A
UNA COMPUTADORA DE VUELO COMERCIAL

MONTERREY, NUEVO LEÓN

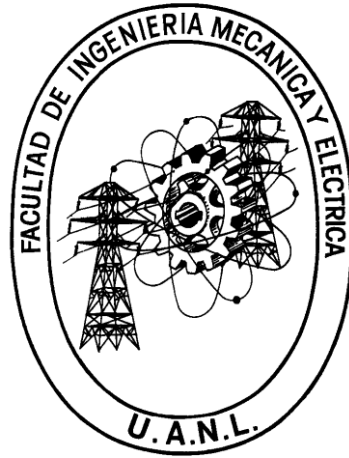
Por

ING. ORLANDO MARTÍNEZ GALVÁN

Como requisito parcial para obtener el Grado de MAESTRÍA EN
INGENIERÍA AERONÁUTICA con Especialidad en DINÁMICA DE
VUELO

JUNIO, 2016

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN
FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA
DIVISIÓN DE ESTUDIOS DE POSGRADO



INTEGRACIÓN DE UN PROCESADOR DIGITAL DE SEÑALES A
UNA COMPUTADORA DE VUELO COMERCIAL

MONTERREY, NUEVO LEÓN

Por

ING. ORLANDO MARTÍNEZ GALVÁN

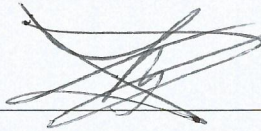
Como requisito parcial para obtener el Grado de MAESTRÍA EN
INGENIERÍA AERONÁUTICA con Especialidad en DINÁMICA DE
VUELO

JUNIO, 2016

Universidad de Autónoma de Nuevo León
Facultad de Ingeniería Mecánica y Eléctrica
Subdirección de estudios de Posgrado

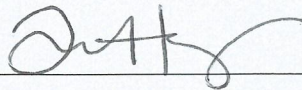
Los miembros del comité de tesis recomendamos que la tesis "INTEGRACIÓN DE UN PROCESADOR DIGITAL DE SEÑALES A UNA COMPUTADORA DE VUELO COMERCIAL" por el Ing. Orlando Martínez Galván con número de Matrícula 1416463 sea aceptada para su defensa como opción a grado de Maestro en Ingeniería Aeronáutica con orientación en Dinámica de Vuelo.

El comité de tesis



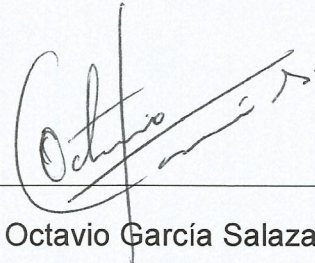
Dr. Luis A. Amézquita Brooks

Asesor



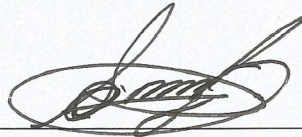
M.C. Diana Hernández Alcántara

Revisor



Dr. Octavio García Salazar

Revisor



Dr. Simón Martínez Martínez

Subdirector de Estudios de Posgrado

San Nicolás de los Garza, Nuevo León, Junio, 2016

Dedicatoria

A mis padres Pascual y Norma por su constante apoyo y consejos que me han brindado en las decisiones que he tomado. A mi hermana, abuelos paternos, abuelos maternos y demás miembros de mi familia que confiaron en mí.

Agradecimientos

Quiero agradecer a mi asesor el Dr. Luis A. Amézquita Brooks por la oportunidad que me ofreció de estudiar esta maestría, por sus enseñanzas en estos casi tres años desde que comencé a trabajar con él, sinceramente muchas gracias.

También un agradecimiento a mis amigos, el Ing. Carlos Santana por ayudarme en el diseño de las bases de prueba y al Ing. Pedro Limón por ayudarme cuando se lo solicité, y a los demás compañeros de generación les agradezco por permitirme formar parte de su grupo durante el primer año.

Por último agradecer a todas aquellas personas anónimas que de manera indirecta me ayudaron durante este ciclo de mi vida.

Tabla de contenido

Resumen	9
1 Introducción.....	11
1.1 Sistemas aéreos no tripulados.....	11
1.1.1 Carga útil	12
1.1.2 Estación terrestre	12
1.1.3 Vehículo aéreo.....	12
1.1.4 Sistema de comunicación	13
1.1.5 Computadora de vuelo	14
1.2 Computadoras de vuelo comerciales.....	15
1.2.1 Configuraciones generales de una computadora de vuelo	18
1.3 Unidades de procesamiento propuestas	19
1.4 Planteamiento del problema.....	20
1.5 Hipótesis	20
1.6 Objetivos de la investigación	21
1.6.1 Objetivo general	21
1.6.2 Objetivos específicos.....	21
1.7 Justificación y uso de resultados	21
1.8 Límites del estudio	22
1.9 Metodología.....	23
2 Plataforma PVTOL.....	25
2.1 Modelado del sistema.....	25
2.2 Banco de pruebas	27
2.2.1 Descripción del funcionamiento de la plataforma PVTOL	29
2.3 Diseño del observador.....	32
2.4 Diseño del controlador.....	35
2.5 Programación del Piccolo F28069 controlSTICK.....	37
2.5.1 Comunicación I2C.....	39
2.5.2 Stateflow chart	48
2.5.3 Condiciones de entrada	53
2.5.4 Estimación de la posición angular	54
2.5.5 Referencias del sistema	55
2.5.6 Controlador.....	59

2.5.7	Modulación por ancho de pulso PWM	60
2.5.8	Script Matlab	63
2.5.9	Monitoreo de variables	65
2.6	Resultados Experimentales	70
3	Cuadri-rotor	75
3.1	Modelado del sistema.....	75
3.2	Banco de pruebas	78
3.2.1	Banco para ángulo de guiñada	78
3.2.2	Banco de pruebas con movimiento en tres ejes	79
3.3	Diseño del observador.....	80
3.4	Diseño de los controladores	82
3.5	Procedimiento de operación.....	90
3.6	Descripción general del sistema	91
3.7	Programación de la computadora de vuelo.	94
3.7.1	Protocolo de comunicación I2C	94
3.7.2	Adquisición y procesamiento de datos	95
3.7.3	Referencias del sistema	102
3.7.4	Implementación de controladores.....	104
3.7.5	Implementación de la ganancia dinámica y matriz de propulsión	104
3.7.6	Escritura por I2C.....	106
3.8	Código para definir las variables a utilizar.....	106
3.9	Código MultiWii 2.2	106
3.10	Resultados experimentales	111
4	Conclusiones.....	116
4.1	Trabajo a futuro	119
4.2	Recomendaciones	119
5	Bibliografía.....	121
	Índice de figuras	123
	Índice de tablas.....	128
6	Anexos	129
6.1	Configuración de Matlab y CCS.....	129
6.1.1	Configuración de Matlab	131
6.1.2	Ejemplo de programación	140

6.1.3	Simulación simultánea.....	149
6.2	GUI Composer.....	152
6.3	Planos	159
6.4	Script de Matlab para el cuadri-rotor.....	164

Resumen

En la actualidad existen computadoras de vuelo de bajo nivel, las cuales están siendo utilizadas en la construcción de pequeños vehículos aéreos del tipo multirotor. Una opción es la Crius_SE v2.5, esta pequeña computadora de vuelo cuenta con un microcontrolador ATmega328 que utiliza la famosa plataforma Arduino y además cuenta con los sensores básicos que conforman una unidad inercial. Sin embargo, esta computadora de vuelo (CDV) muestra una significativa deficiencia al realizar cálculos de funciones complejas ya que el microcontrolador es de bajo nivel. Es por eso que se trata de implementar un microcontrolador de nivel medio, el cual se encargaría de realizar los cálculos de las ecuaciones de control, el sistema de navegación y el piloto automático. Este microcontrolador es el Piccolo F28069 controlStick de Texas Instruments, el cual tiene la facilidad de operar números de punto flotante, lo cual lo hace muy atractivo para realizar los cálculos mencionados anteriormente.

Para probar las capacidades del Piccolo se implementa un sistema tipo PVTOL con el cual se realizaron pruebas dentro y fuera del túnel de viento. Los resultados obtenidos son favorables y demostraron el efecto de las ráfagas de viento sobre el sistema de control.

La integración del Crius con el Piccolo se realizó tomando en cuenta un cuadricóptero. Durante ésta integración se modifica el código del Crius eliminando los algoritmos de control y sustituyéndolos por bloques de comunicación I2C. En el caso del Piccolo se toma como base el código para el sistema PVTOL.

Como la aplicación principal es la implementación sencilla de controladores experimentales, se utilizaron dos plataformas de pruebas para ajustar los controladores. Una plataforma permite movimiento únicamente en el eje z, mientras que la otra plataforma tiene libertad en los ejes x, y, z. Después de validar los controladores en las plataformas se realizaron los vuelos libres.

En el capítulo 1 se encuentran los objetivos de esta tesis, así como la hipótesis y metodología a seguir. El modelado, programación, implementación y resultados del sistema PVTOL se encuentran en el capítulo 2. El capítulo 3, que corresponde al cuadri-rotor, incluye el modelado del sistema, diseño de observador y controlador, programación de las unidades de procesamiento utilizadas en esta tesis y los resultados experimentales en una plataforma experimental y los de vuelo libre. Por último las conclusiones se pueden encontrar en el capítulo 4, además de algunas recomendaciones y trabajo a futuro propuesto. En la sección de anexos se encuentra información que complementa el contenido de esta tesis, como es la configuración de los softwares utilizados, algunas características de interés que poseen estos softwares, planos de las piezas que utilizan las plataformas experimentales y una parte de código para definir las variables y formar los controladores.

1 Introducción

Actualmente los vehículos aéreos no tripulados (VANT) tienen muchas aplicaciones. Estos vehículos se pueden clasificar de manera general como ala fija o ala rotativa. Sin embargo también se pueden clasificar por el tamaño: vehículo aéreo no tripulado, mini VANT, micro VANT y nano VANT.

Los crecientes avances de la tecnología, han permitido tener acceso a dispositivos que antes estaban restringidos al público en general, como es el caso de los sensores tipo MEMS (Micro electrical mechanic sensors, por sus siglas en ingles). Este tipo de encapsulado comúnmente es utilizado en el desarrollo de computadoras de vuelo comerciales. Otro caso es el avance que hay en los microcontroladores y en la accesibilidad en tarjetas embebidas que están en constante mejora y abundan en el mercado.

1.1 Sistemas aéreos no tripulados

Un sistema aéreo no tripulado (SANT) consta de todos los elementos involucrados en la operación de un VANT. Un SANT puede estar compuesto de:

- Carga útil
- VANT
- Estación de monitoreo terrestre
- Plataforma de lanzamiento, recuperación y traslado de todos los elementos
- Computadora de vuelo.

1.1.1 Carga útil

Elementos que son necesarios para cumplir las necesidades de una misión que tenga el VANT según la aplicación requerida. Por ejemplo, si se va a vigilar o monitorear una zona ambiental a través de una cámara, ésta es la carga útil.

1.1.2 Estación terrestre

Es el centro de control mediante el cual el usuario se comunica con el vehículo para controlarlo o manipular la carga útil a través de una interfaz gráfica. Esta interfaz puede mostrar información de los sensores a bordo y de la carga útil instalada según la aplicación que tenga.

1.1.3 Vehículo aéreo

El vehículo forma parte del sistema, puede ser de ala fija o ala rotativa. Cuando el vehículo no es tripulado, quiere decir que la tripulación junto a los componentes que utilizan, por ejemplo la cabina, los controles, instrumentación, entre otros, son remplazados por una computadora de vuelo y un sistema de control.

Las aeronaves no tripuladas se pueden clasificar de distintas maneras ya sea por su tamaño, rango de operación, entre otras.

Antes de continuar con la siguiente sección es preciso aclarar la diferencia entre una aeronave no tripulada, una de aeromodelismo y un "Drone" que comúnmente son confundidos.

Un vehículo de aeromodelismo es operado con un radiocontrol dentro del rango de visión del operador. Cuenta con una cantidad limitada de instrucciones para operar el vehículo. Por ejemplo, hacerlo ascender, descender, moverlo a la derecha o a la izquierda.

Un Drone puede volar fuera del rango de visión del operador, pero no puede tener interacción con este. Este vehículo tiene pre-programada la misión a realizar, así

como la ruta desde que despegar hasta que aterriza y sólo cuando termina la misión el usuario puede obtener la información almacenada del vehículo.

Un vehículo aéreo no tripulado, al igual que el dron, puede volar fuera del rango de visión del operador, sin embargo tiene cierto grado de inteligencia y puede ser capaz de interactuar con el operador, es decir, recibe instrucciones o envía información de la carga útil, por ejemplo la imagen de una cámara térmica, así como información relevante como posición, velocidad, orientación y altitud. [1]

1.1.4 Sistema de comunicación

El vehículo necesita un módulo de comunicación con la estación terrestre. Este módulo de comunicación transmite y recibe información del vehículo a la estación terrestre y viceversa. Generalmente la comunicación se realiza por medio de señales de radio aunque los sistemas más avanzados pueden utilizar otra tecnología. Los enlaces de comunicación son:

- Enlace ascendente
- Enlace descendente.

El enlace ascendente es la comunicación desde la estación terrestre hacia la aeronave. Durante esta comunicación usualmente se transmiten señales de control de la aeronave (en caso de ser necesario), y señales para operar la carga útil.

El enlace descendente se realiza cuando el vehículo aéreo transmite información a la estación terrestre. La información de interés que transmite es la posición actual de la aeronave, información recopilada de los sensores, y además la información de la carga útil.

1.1.5 Computadora de vuelo

Este dispositivo es el que recopila la información de los sensores a bordo, las señales de radio de la estación terrestre, manipula las superficies de control o elementos de control (en caso de que el vehículo tenga) y ejecuta los algoritmos de software del sistema.

Una computadora de vuelo (CDV) de un VANT puede presentar cierto grado de autonomía, desde 0 hasta 100% que la hace completamente autónoma.

Se puede definir el concepto autonomía para una computadora de vuelo como: la habilidad para realizar una misión sin la intervención de un operador.

Al presentar algún porcentaje de autonomía se puede decir que la CDV tiene un piloto automático. El nivel de autonomía que presente la CDV determina los procesos que el piloto automático va a realizar, es decir, desde una tarea sencilla como es mantener los ángulos de guiñada, alabeo y cabeceo, además de la velocidad y altitud del vehículo, hasta ser capaz de despegar, seguir una ruta de vuelo pre-programada y aterrizar, todo sin que el usuario intervenga.

La Figura 1.1 muestra una arquitectura de software de una CDV autónoma donde:

- El vehículo tiene como entradas los esfuerzos de control y las perturbaciones del viento o alguna otra y mide la salida a través de sensores a bordo.
- El piloto automático (“autopilot”) consta de algoritmos de control de bajo nivel los cuales sólo mantiene las posiciones angulares de alabeo, cabeceo y guiñada, altitud y velocidad del vehículo.
- El bloque del seguimiento de trayectoria (“path following”) mantiene al vehículo en la trayectoria definida.
- El siguiente bloque llamado gestor de trayectoria (“path manager”) tiene los algoritmos para generar la trayectoria con líneas rectas y parábolas con posición y velocidad a partir de “waypoints”.
- El bloque superior planificador de trayectoria es en donde se define la trayectoria a realizar por medio de “waypoints”.

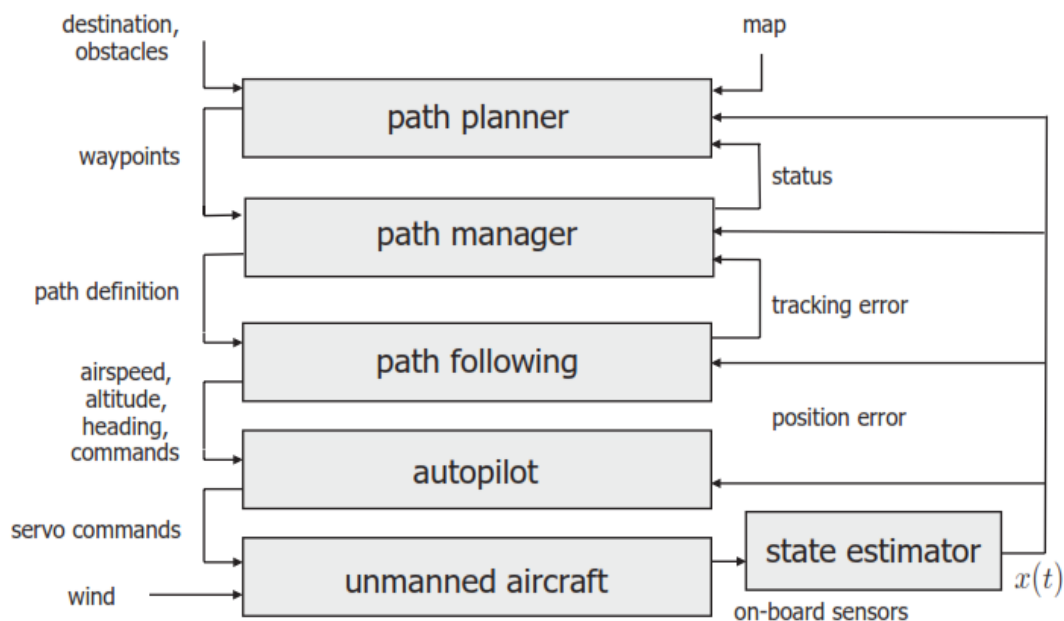


Figura 1.1 Arquitectura del sistema de una CDV autónoma [2]

Tomando como ejemplo la Figura 1.1 se puede apreciar que se utiliza un estimador de estados u observador para estimar variables que no es posible medir directamente por medio de los sensores.

1.2 Computadoras de vuelo comerciales

En la actualidad existen diversas CDV que ya cuentan con su interfaz gráfica para la estación en tierra que despliega información de las mediciones de los sensores, orientación del vehículo, altitud, etc. A simple vista no parece que este tipo de CDV tenga algún problema ya que al ser probadas funcionan bien, pero al analizar de forma detallada el código se encuentra que:

- Las mediciones de los sensores y las operaciones que realiza en algunas ocasiones no son las adecuadas.
- El periodo de muestreo no es constante y la variación que tiene está fuera de un rango aceptable para este tipo de sistema.

Sin embargo algunas de las ventajas que tienen las CDV comerciales son:

- La facilidad con la que se pueden configurar dependiendo del tipo de vehículo.
- Ofrecen la libertad de escoger el modelo de los sensores a utilizar.
- Cuentan con una interfaz gráfica para una estación terrestre.

Como las CDV están hechas para el público en general, normalmente los controladores que utilizan son del tipo PID, por lo que el usuario sólo tiene que ajustar las ganancias de estos controladores por medio de prueba y error hasta que el vehículo esté lo suficientemente estable para el usuario. En la Tabla 1-1 se muestra una comparación de algunas CDV disponibles en el mercado.

Tabla 1-1 Comparación de CDV comerciales

CDV	Parrot AR Drone	Piccolo SL	Ardupilot	Pixhawk	Crius SE 2.5	Kestrel 2.2	Piccolo F28069
Tamaño (cm)	7.2*7.8*1.6	13.1*5.56*1.9	4*6.5*1	5*8.15*1.55	4*4*1.16	5.08*3.5*1.2	5.5*2.4*1.6
Peso (gr)	N/A	124	N/A	38	9.3	16.7	N/A
Consumo de energía	N/A	N/A	500mA	N/A	N/A	500mA	N/A
Código abierto	No	No	Si	Si	Si	No	Si
Fabricante	Parrot DSP de video TMS320DMC64x a 800 MHz con un microprocesador ARM Cortex A8 de 32 bits a 1 GHz	CloudCap	DIY Drones Team Microcontrolador ATmega2560P de 8 bits a 16MHz	3D Robotics Microcontrolador STM32F4S7 Cortex M4 de 32 bits a 168 MHz con FPU	N/A Microcontrolador Atmega328P de 8bits a 16 MHz	Lockead Martin RCM 3400 con microprocesador Rabbit 3000 de 8 bits a 29 MHz	Texas Instruments Microcontrolador TMS320C28x de 32 bits a 90 MHz con FPU
Desempeño	Acelerometro 3 ejes, giroscopio 2 ejes, magnetometro, sensor ultrasonico	Sensor de presion 11-115 KPa, giroscopio de 3 ejes, acelerometro 3 ejes, GPS 4Hz	Sensor de presion barometrica, magnetometro, acelerometro y giroscopio de 3 ejes cada uno	2 giroscopios de 3 ejes, 2 acelerometros de 3 ejes, magnetometro de 3 ejes, barometro	Acelerometro de 3 ejes, giroscopio de 3 ejes, magnetometro de 3 ejes, sensor de presion barometrica	Giroscopio de 3 ejes, acelerometro de 3 ejes, magnetometro 2 ejes, sensor de presion absoluta	N/A
Sensores							

1.2.1 Configuraciones generales de una computadora de vuelo

Existen diversas opciones para desarrollar una CDV, por ejemplo se pueden clasificar como:

- CDV con una unidad de procesamiento de alto nivel.
- CDV con dos unidades de procesamiento.
- CDV con más de dos unidades de procesamiento.

La primera opción utiliza una unidad de procesamiento de alto nivel donde se desarrolla todo el sistema de software de la CDV, incluye los algoritmos del sistema de navegación y comunicaciones. Sin embargo, en el estudio del estado del arte realizado en esta tesis se encontró que la cantidad de reportes recientes que toman esta opción es menor en comparación con las otras opciones. Por ejemplo, de la Tabla 1-1 el Crius SE 2.5, el Ardupilot Mega y el Kestrel 2.2 sólo utilizan un microcontrolador para todo el sistema.

La segunda opción utiliza dos unidades de procesamiento con microcontroladores o microprocesadores de igual o diferente nivel, en donde generalmente uno se encarga de todas las funciones del sistema como los algoritmos de navegación (sólo adquisición de datos), el piloto automático y realizar la comunicación entre las dos unidades, con la estación en tierra, motores y control remoto. La otra unidad de procesamiento es utilizada como parte de la IMU, es decir, adquiere la información de los sensores y el GPS (en caso de contar con uno) y realiza las estimaciones a partir de los sensores entregando a la unidad principal los resultados de la adquisición. Por ejemplo, de la Tabla 1-1 el Parrot AR Drone utiliza un PIC24HJ16GP304 de 16 bits a 40 MHz como parte de la IMU además del microprocesador principal. El Pixhawk también utiliza dos unidades de procesamiento, la mencionada en la tabla anterior y la STM32F103 de 32 bits. En [3] se seleccionó el piloto automático Kestrel 2.4 como sistema de navegación pero la computadora principal es la VxWorks 5.0.

La tercera opción consiste en utilizar más de dos unidades de procesamiento donde cada una realiza una tarea específica, por ejemplo que una unidad se

encargue de la adquisición y procesamiento de los datos de los sensores, otra se encargue de establecer las comunicaciones con la estación en tierra y otra que sea la unidad principal la cual mantiene el orden de ejecución del sistema, con el fin de optimizar los algoritmos del sistema de la CDV final. En [4] se propone utilizar la Gumstix Verdex Pro como plataforma de control y la MNAV100CA como IMU, también utiliza la Gumstix Overo Fire como plataforma de visión ya que utilizan una cámara para navegación en interiores. El piloto automático AP04 fue utilizado en [5], esta plataforma incluye funciones como despegue y aterrizaje automático, trayectoria de seguimiento de puntos. La computadora PC/104 es utilizada en [6] para ejecutar el sistema de control utilizando la información que recibe de otros módulos.

En las opciones no se toman en cuenta las unidades de procesamiento dedicadas a la carga útil, sólo a las encargadas a mantener un vuelo estable y la navegación.

1.3 Unidades de procesamiento propuestas

El Crius SE 2.5 es de bajo costo y es muy utilizada por aficionados que desean un vehículo aéreo tipo multi-rotor, como por ejemplo un cuadri-rotor. Dentro de esta tarjeta como lo muestra la Tabla 1-1 se encuentra un ATmega328 el cual se considera de bajo nivel al ser de 8 bits, esto quiere decir que puede realizar una operación con datos que tengan el tamaño de 8 bits (0 a 255 en escala decimal); aunque también puede realizar operaciones de 16 o 32 bits, pero para poder llevarlas a cabo las hace en partes de 8 bits cada operación lo cual es un punto negativo ya que el hacer este tipo de cálculo le toma un poco más de tiempo y se reduce el desempeño. Los sensores con los que cuenta el Crius SE 2.5 son utilizados para nivelar al vehículo, saber su orientación y estimar la altitud en la que éste se encuentra.

Como las operaciones que utilizan el sistema de navegación y los controladores pueden incluir operaciones no lineales (coseno, seno, tangente), los resultados pueden contener números con decimales que son considerados números de

punto flotante y adquieren un tamaño de 32 bits con el fin de mantener la exactitud. Es por eso que se busca un MCU que:

- pueda manejar operaciones directas de números con 32 bits
- que cuente con un programador integrado en una tarjeta electrónica
- que no genere mucho peso adicional al vehículo y que sea compacta.

La unidad seleccionada es el Piccolo F28069 controlStick de Texas Instruments la cual tiene un MCU de 32 bits que pertenece a la familia C2000. Lo interesante de este MCU, es que tiene una unidad de punto flotante (FPU, Floating point unit), esto es, que maneja éste tipo de números a nivel de hardware. Este modelo se utiliza como un procesador digital de señales (DSP, por sus siglas en ingles).

1.4 Planteamiento del problema.

Debido al auge que han tenido los micro vehículos aéreos no tripulados (MVANT's) de configuración multi-rotor en los últimos años, existen en el mercado diversas computadoras de vuelo que utilizan MCU de bajo nivel, pero que están respaldadas por una gran comunidad que constantemente realiza mejoras. Sin embargo, al utilizar un MCU de bajo nivel, en el código se tienen preferencia las comunicaciones con sensores, cámaras, etc., y la parte del sistema de control está descuidada, es decir, en las actualizaciones que se han realizado esta parte no se ha tenido grandes mejoras.

1.5 Hipótesis

Una computadora de vuelo autónoma o semiautónoma de alto rendimiento y de bajo costo se puede implementar a partir de una computadora de vuelo comercial de bajo nivel y un procesador digital de señales de nivel superior. Ambos con lenguajes de programación diferentes.

1.6 Objetivos de la investigación

1.6.1 Objetivo general

Diseñar e implementar una computadora de vuelo basada en una plataforma de código abierto complementada con un DSP de Texas Instruments para mejorar las capacidades de procesamiento y facilitar la implementación de los esquemas de control del piloto automático.

1.6.2 Objetivos específicos

Adaptar una plataforma de piloto automático de código abierto a un DSP de Texas Instruments. En particular se partirá de la CDV "Crius_SE" en su versión 2.5 para aprovechar su plataforma de adquisición de datos de los sensores y se mejorará el sistema de control utilizando un MCU de mayor nivel.

Lograr la sincronización de dos MCU de diferentes características para realizar una CDV de alto rendimiento.

Realizar la programación del Piccolo F28069 mediante Simulink y Matlab utilizando la herramienta Stateflow para ejecutar los controladores y operaciones no lineales.

Que este sistema sea modular, de tal forma que sea fácil programar o modificar (cambiar de controlador, ganancias, etc.).

Diseñar los controladores para cabeceo, alabeo, guiñada y altitud a partir del modelo matemático de un cuadri-rotor.

Incluir un sistema de navegación inercial.

Validación en plataforma experimental.

1.7 Justificación y uso de resultados

Comercialmente las computadoras de vuelo de alto nivel que pueden operar de forma autónoma son costosas, como lo muestra la comparación hecha en [7]. Además algunas no son de código abierto, lo que introduce la necesidad de

desarrollar pilotos automáticos que sean de código abierto, de bajo costo y fáciles de programar.

Al realizar una investigación preliminar, que consistió en determinar el tiempo que tarda el MCU del Crius en realizar algunas tareas comunes en comparación con el MCU del DSP de Texas Instruments, los resultados muestran que al realizar cálculos con números de punto flotante el Crius es cuatro veces más lento que el DSP. Al analizar los tiempos que tardan ambos en realizar las comunicaciones, se encuentra que el Crius es 20% más rápido para transmitir datos por comunicación Serial. Por otro lado, la interfaz de usuario de la estación en tierra que utiliza el Crius mantiene la comunicación por Serial. Por lo cual se pretende utilizar esta interfaz gráfica de usuario para aprovechar algunas de sus características, como la opción que tiene para almacenar información.

Este sistema tendrá como finalidad de disponer de una CDV en la que sea posible manipular de una forma sencilla los controladores al momento de cambiarlos o modificarlos. Así como una base para trabajos futuros en el desarrollo de MVANT's que necesiten una CDV pequeña y de poco peso.

1.8 Límites del estudio

Como se ha mencionado anteriormente esta investigación partirá de una CDV comercial ya que para realizarla desde cero se necesitaría de más tiempo.

No se desarrollará la interfaz de la estación en tierra ya que se partirá de una CDV comercial que ya posee este módulo.

El análisis y diseño de los controladores se realizarán en tiempo continuo, pero al incluirlas en el código se tendrán que trasladar a tiempo discreto. El desarrollo de los controladores tiene como objetivo el probar la CDV. Es decir, no es una de las aportaciones principales de la tesis.

Por último, no se toman en cuenta el diseño de la estructura de la plataforma experimental, ni de las hélices.

1.9 Metodología

Tipo y diseño general del estudio

- 1.- Realizar durante toda la investigación una revisión bibliográfica en distintos medios como libros o artículos relacionados al tema.
- 2.- Realizar la ingeniería inversa a la CDV Crius. Esto consiste en revisar el código de programación, que es abierto, para identificar las partes necesarias que mantengan el funcionamiento sin errores. Además para identificar las secciones innecesarias que al ser eliminadas no afecten al sistema.
- 3.- Programar el DSP de Texas Instruments. Para poder programar por medio de Simulink/Matlab, lo primero que hay que hacer es configurar Matlab para que pueda generar el código que utiliza el software Code Composer Studio (CCS). Por medio de Simulink/Matlab se pretende generar el código y el CCS es sólo para programar al DSP. Al principio sólo se programan cosas sencillas para familiarizarse con el método de programación y saber que funciones de Simulink son soportadas.
- 4.- Utilizar Stateflow. Se pretende utilizar esta herramienta de Simulink para simplificar los códigos hechos con el fin de facilitar la programación.
- 5.- Monitorear variables con el Code Composer Studio (CCS). Identificar las interrupciones internas del DSP que son necesarias para utilizar los protocolos de comunicación aprovechando la capacidad que tiene el CCS de monitorear en tiempo real variables. Utilizar esta información en la programación avanzada, en la cual se contempla programar el DSP utilizando algunas interrupciones internas y diferentes protocolos de comunicación con la ayuda de la interacción con el CCS. Así como mejorar el uso del Stateflow.
- 6.- Comunicación entre los dos dispositivos. Realizar prácticas en las cuales interactúen por medio del protocolo de comunicación I2C en sus dos modos de configuración, maestro-esclavo, esclavo-maestro.

7.- Desarrollo del sistema de navegación. Para el diseño del sistema de navegación, primero se programan los sensores para tomar mediciones y procesar dicha información. Después utilizando las mediciones de los sensores se programan los algoritmos para estimar la posición y velocidad, así como los algoritmos para corregir o eliminar la deriva que se puede generar.

8.- Diseño de los controladores. El diseño de los controladores es tomando como base al modelo matemático de un cuadri-rotor.

9.- Desarrollar y programar el piloto automático. Cuando se tenga el sistema de navegación y los controladores listos se puede pasar a la siguiente etapa la cual consiste en desarrollar y programar el piloto automático. Cabe destacar que los controladores pueden ser modificados según las características deseadas.

10.- Integrar el DSP a la CDV. Primero que se tiene que optimizar el código del Crius. Esta optimización es para eliminar las funciones innecesarias. La integración tiene como función compartir información y cerrar el lazo del sistema de la CDV.

11.- Visualización en la estación en tierra. Aquí lo que se busca es que una vez terminada la CDV no tenga ningún problema en comunicarse con la interfaz de usuario. Si existe algún error se tiene que corregirlo sin afectar mucho el sistema.

12.- Validación de la CDV con el DSP. Una vez terminado el diseño de la CDV se tendrá que evaluar su funcionamiento en alguna plataforma experimental, como por ejemplo: PVTOL, cuadri-rotor, etc. Así como su desempeño con la interfaz gráfica de la estación terrestre.

2 Plataforma PVTOL

2.1 Modelado del sistema

Un PVTOL puede ser considerado como una versión simplificada de un vehículo multi-rotor simétrico con reducidos grados de libertad. Sin embargo, se puede utilizar como un paso inicial en el desarrollo de un vehículo multi-rotor con una configuración más compleja.

El modelo del PVTOL, que se muestra en la Figura 2.1, tiene tres variables de salida que corresponden a la posición y a la orientación (x, y, ϕ) y dos entradas de control que son el empuje y el momento generados por el empuje de los motores (f_1, f_2) . La distancia entre el centro de gravedad del vehículo y los motores es dada por l . Las fuerzas de entrada f_1 y f_2 producen una fuerza acumulativa sobre el centro de la masa del vehículo igual a:

$$u_1 = f_1 + f_2 \quad (1)$$

Y un momento acumulativo igual a:

$$u_2 = (f_2 - f_1)l \quad (2)$$

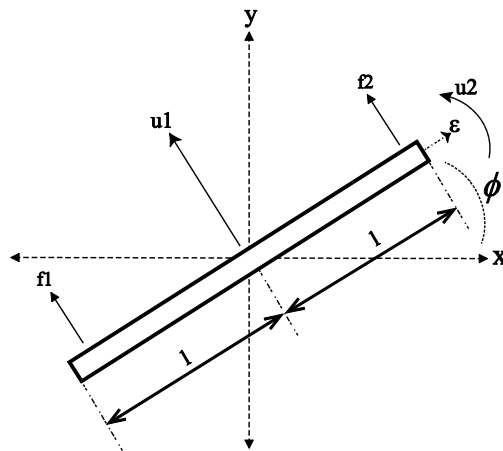


Figura 2.1 Modelo del PVTOL (Planar Vertical Takeoff and Landing)

Para variaciones pequeñas de velocidad el empuje de la hélice i puede ser aproximada por:

$$f_i = \frac{1}{2} \rho S C_T k V_i \quad (3)$$

Donde ρ es la densidad del aire, S el área circular de la hélice, C_T el coeficiente de levantamiento de la hélice y kV_i es la velocidad angular de la hélice del motor i con el voltaje V_i .

La aproximación de (3) omite el estado transitorio del motor ya que se asume que es más rápido que las dinámicas del sistema. A excepción del voltaje del motor V_i , los demás elementos de f_i pueden ser considerados constantes. Por lo tanto:

$$f_i = k_p V_i \quad (4)$$

Donde la constante k_p puede ser caracterizada experimentalmente para cada motor según la hélice que tenga. El voltaje V_i normalmente es aplicado por medio de un inversor de modulación de ancho de pulso (PWM).

Considerando las ecuaciones (1) y (2), el modelo matemático de este sistema están dadas por las siguientes ecuaciones:

$$\begin{aligned} m\ddot{x} &= u_1 \sin \phi - \varepsilon u_2 \cos \phi \\ m\ddot{y} &= -u_1 \cos \phi - \varepsilon u_2 \sin \phi - mg \\ J\ddot{\phi} &= u_2 \end{aligned} \quad (5)$$

En donde ε es un coeficiente muy pequeño que caracteriza el acoplamiento entre el momento de alabeo u_2 y la aceleración lateral que tiene el vehículo, m es la masa del vehículo, J es el momento de inercia y g es la aceleración gravitacional [8] [9].

2.2 Banco de pruebas

El banco de pruebas que se muestra en la Figura 2.2 está equipado con:

- Dos motores sin escobillas (brushless) de 980kv (donde kv = revolución por volt) con hélices de 8x4, que significa que tiene 8 pulgadas de diámetro y un ángulo de pala de 4°.
- Dos controladores electrónicos de velocidad (ESC, Electronic Speed Control) que soportan una corriente de 30 A.
- Una unidad de medición inercial (IMU) de bajo costo con el sensor MPU-6050 que consiste en un acelerómetro de 3 ejes y un giroscopio de 3 ejes
- Un microcontrolador de bajo costo de la familia C2000 de Texas Instruments.
- Transmisor y receptor inalámbrico.
- Rodamientos de baja fricción y una estructura de aluminio de alta resistencia.

Todo el sistema de aviónica está preparado para funcionar con una fuente de poder estacionaria o con una batería tipo Li-Po en caso de ser necesario operar el sistema a distancia.

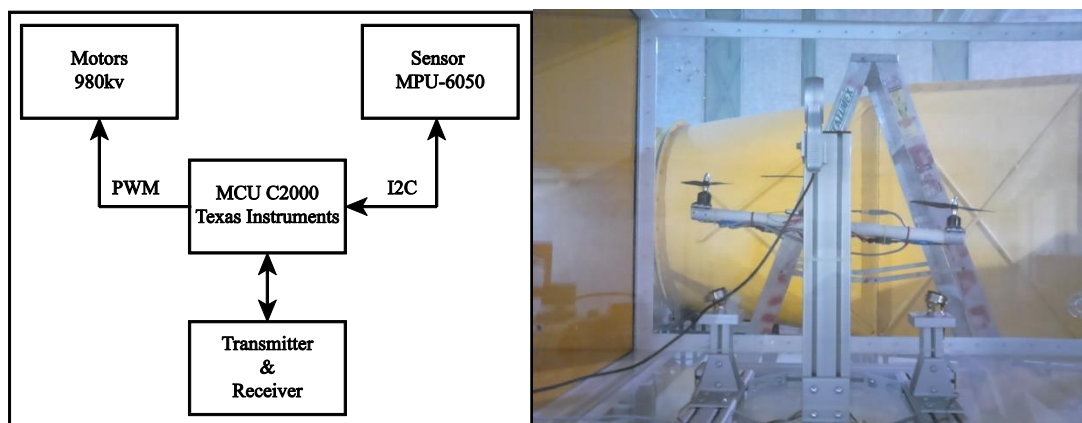


Figura 2.2 Banco de pruebas

Mientras que la configuración de la propulsión de este banco de pruebas es similar al de un PVTOL tradicional, sólo permite el movimiento de alabeo. Por lo tanto, las primeras dos ecuaciones de (5) se desprecian; además, debido a la fricción de los baleros la última ecuación de (5) debe de ser modificada. En consecuencia, considerando sólo el movimiento de alabeo y la fricción de los baleros (identificada como b), el modelo resultante es:

$$J\ddot{\phi} = (f_2 - f_1)l - b\dot{\phi} \quad (6)$$

Como la ecuación (6) está totalmente actuada, la estrategia que se plantea para activar a los dos motores con una sola variable es:

$$\begin{aligned} V_1 &= \left(\frac{u_c + 1}{2} \right) \\ V_2 &= \left(\frac{-u_c + 1}{2} \right) \end{aligned} \quad (7)$$

donde u_c es la nueva entrada de control normalizada, esto significa que cuando $u_c=0$ el empuje se mantiene en 50% y el mayor momento posible positivo o negativo es generado con $u_c=1$ o $u_c=-1$ respectivamente.

Considerando las ecuaciones (6) y (7) es posible calcular la función de transferencia para las dinámicas de alabeo como:

$$g_{\phi}(s) = \frac{\phi(s)}{u_c(s)} = \frac{lk_p}{(Js + b)s} \quad (8)$$

Mediante un proceso de identificación experimental utilizando la estructura de (8) se obtuvo el siguiente modelo:

$$g_{\phi}(s) = \frac{903.5}{s(s + 0.02857)} \quad (9)$$

La Figura 2.3 muestra una comparación entre la respuesta experimental del banco de prueba con el modelo (9). Esta respuesta se obtuvo en lazo cerrado utilizando el desarrollo del controlador descrito en la sección 2.4. Esto es debido al hecho de que la comparación en lazo abierto difiere porque el modelo (9) es críticamente

estable. Finalmente, un alto grado de incertidumbre fue detectado en el parámetro de la fricción viscosa b . Además, el fenómeno stick-slip también fue detectado. Este hecho se discute en la sección 2.4.

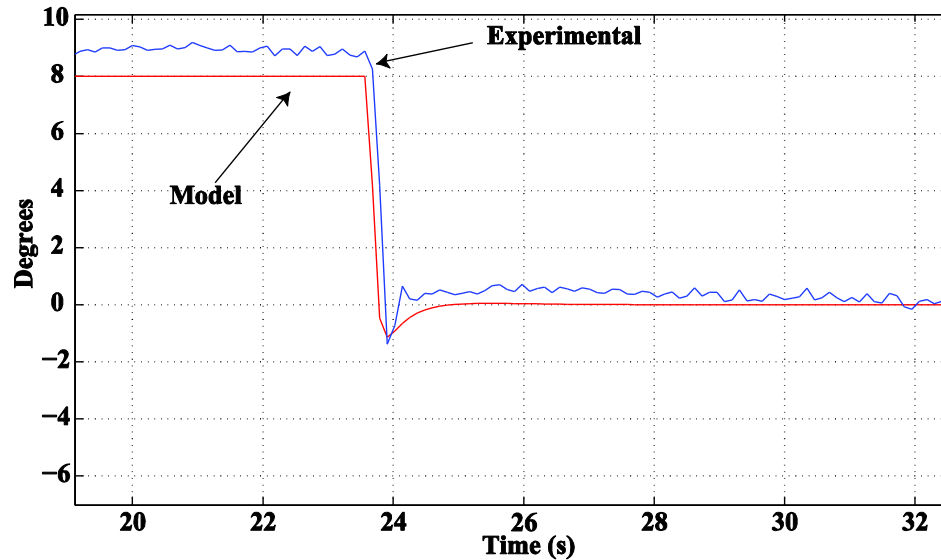


Figura 2.3 Comparación entre la respuesta en lazo cerrado experimental y el modelo

2.2.1 Descripción del funcionamiento de la plataforma PVTOL

El Piccolo F28069 controlSTICK controla a los motores por medio de señales PWM, se comunica con los sensores por medio del protocolo de comunicación I2C, realiza los algoritmos de control del sistema y además tiene la capacidad de generar dos tipos de referencias internas que consisten en una serie de escalones y una señal tipo Chirp de 0.05Hz a 1Hz. Además el Piccolo cuenta con la opción de utilizar una referencia externa generada por el radiocontrol. Los sensores con los que se trabajan son dos: acelerómetro de 3 ejes y giroscopio de 3 ejes cuya función es medir la posición angular del alabeo en grados, el sensor utilizado es el MPU6050. Como medida de seguridad cuenta con dos interruptores que al activarse cualquiera el sistema se detiene y estos mismos sirven para reiniciar el sistema PVTOL.

El diagrama esquemático se muestra en la Figura 2.4. Se puede apreciar que existen tres voltajes diferentes: 11.1V, 5V y 3.3V.

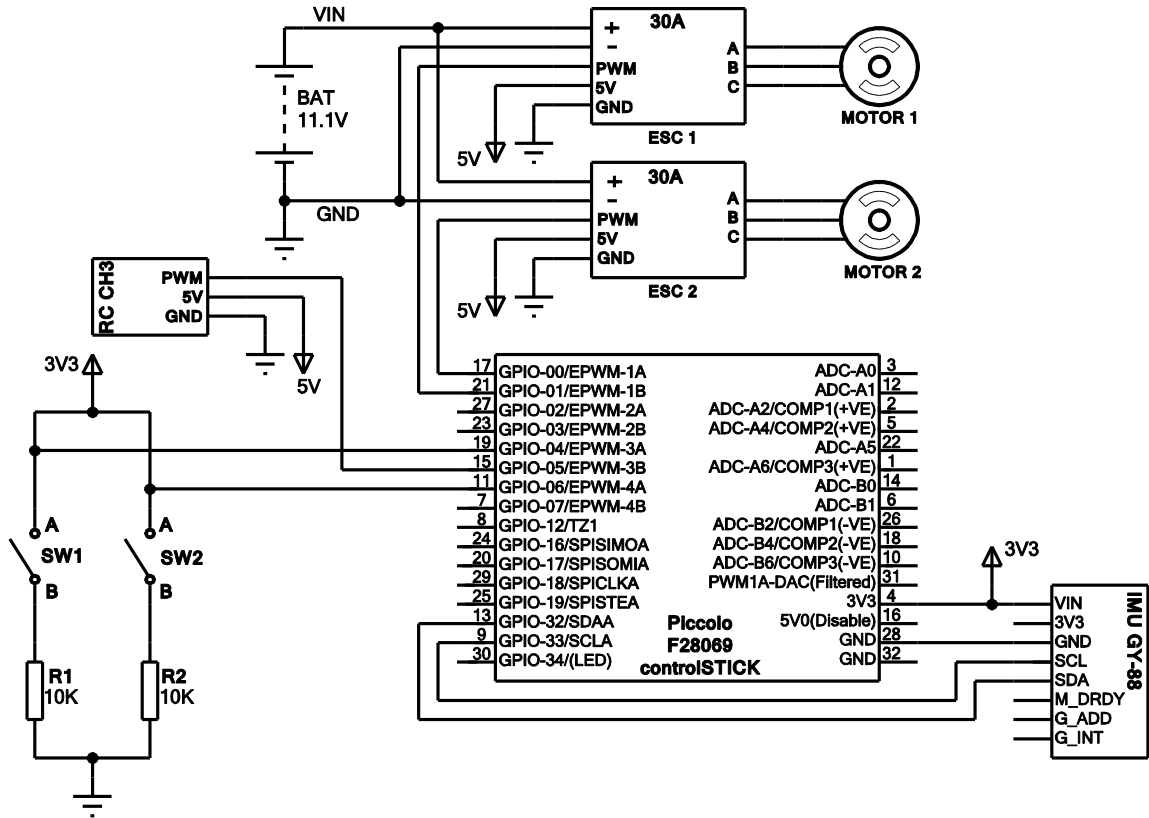


Figura 2.4 Diagrama esquemático

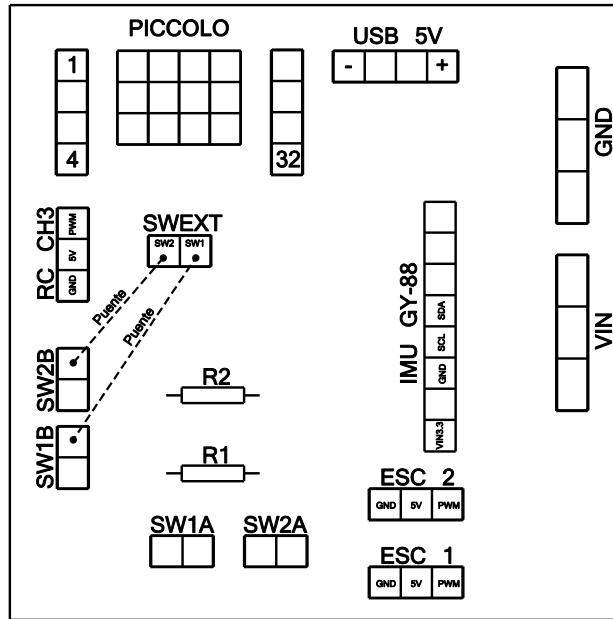


Figura 2.5 Placa de conexiones PCB

Tabla 2-1 Descripción de puertos de conexión

Simbología	Descripción
Piccolo	Microcontrolador de 32 pines
USB 5V	Conector para alimentar con 5v al Piccolo cuando no se requiera la pc.
RC CH3	Canal 3 del receptor
SW1A/SW1B	Interruptor de lado izquierdo
SW2A/SW2B	Interruptor de lado derecho
SWEXT	Puerto de enlace entre los interruptores y el Piccolo
R1,R2	Resistencias Pull-Down de 10K
IMU GY-88	Central inercial que contiene los sensores
VIN	Puerto positivo de la fuente o batería a los ESC
GND	Puerto común de la fuente o batería a los ESC
ESC 1	Controlador electrónico de velocidad del motor 1
ESC 2	Controlador electrónico de velocidad del motor 2

En la sección 2.5 se hace una descripción a fondo de la programación de la computadora de vuelo.

2.3 Diseño del observador

La unidad de medición inercial (IMU) usada en los experimentos tiene acceso a las mediciones del giroscopio y acelerómetro. Esto permite implementar una combinación de ambos sensores para incrementar el rechazo a perturbaciones externas. En particular, se observó que las mediciones del acelerómetro son muy sensibles a las vibraciones de alta frecuencia producidas por los motores y por las ráfagas de viento.

El ángulo de alabeo se puede obtener a través de las mediciones del acelerómetro despreciando las aceleraciones traslacionales (esto es, $\ddot{x}=0$ y $\ddot{y}=0$) y considerando sólo la gravedad. Esta suposición es normalmente valida en el estado estacionario cuando el vehículo ha ganado una velocidad constante. En este caso si $\phi \in (-90^\circ, 90^\circ)$ entonces el ángulo de alabeo puede ser estimado como:

$$\phi_a = \tan^{-1} \left(\frac{a_x}{a_y} \right) \quad (10)$$

Donde a_x y a_y son las mediciones del acelerómetro en los ejes x y y respectivamente.

Ya que los acelerómetros están expuestos al ruido de alta frecuencia debido a las vibraciones y a las aceleraciones generadas por el movimiento transitorio del vehículo, ϕ_a es considerado con menos error a bajas frecuencias y con más error en altas frecuencias.

Por otro lado, las mediciones del giroscopio pueden ser utilizadas para estimar el ángulo de alabeo con la integración numérica directa de la medición:

$$\phi_g = \int \omega_g dt \quad (11)$$

Donde ω_g es la medición del giroscopio.

Es bien conocido que la estimación de la posición angular por medio de la integración directa de las mediciones del giroscopio produce deriva, la cual es producida por la integración de las componentes del ruido de baja frecuencia de las mediciones del giroscopio de ω_g . Por lo tanto, ϕ_g se considera limpio en altas frecuencias y ruidoso en bajas frecuencias. En la siguiente figura se muestra la combinación de ambas estimaciones por medio de un observador de Luenberger.

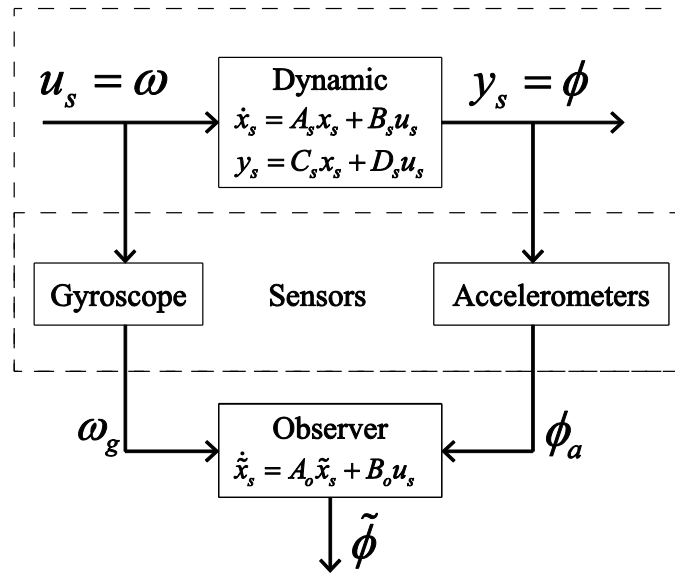


Figura 2.6 Observador basado en la mezcla de sensores

La relación dinámica entre ϕ_a y ω_g sin perturbaciones es dado por $\dot{\phi}_a = \omega_g$, así un observador de Luenberger para estimar ϕ se puede obtener considerando el siguiente sistema lineal:

$$\begin{aligned} \dot{x}_s &= A_s x_s + B_s u_s \\ \dot{y}_s &= C_s x_s + D_s u_s \end{aligned} \quad (12)$$

Con $A_s = 0, B_s = 1, C_s = 1, D_s = 0, u_s = \omega$ y $x_s = \phi$

Utilizando las ecuaciones clásicas de Luenberger el observador es dado por:

$$\dot{\tilde{x}} = A_o \tilde{x}_s + B_o u_s \quad (13)$$

Con $A_o = A_s - HC_s, B_o = B_s - HD_s$ donde $\tilde{x}_s = \tilde{\phi}$ es la estimación del ángulo de alabeo. La ganancia del observador H fue diseñada para que el polo del observador estuviera en -0.4 .

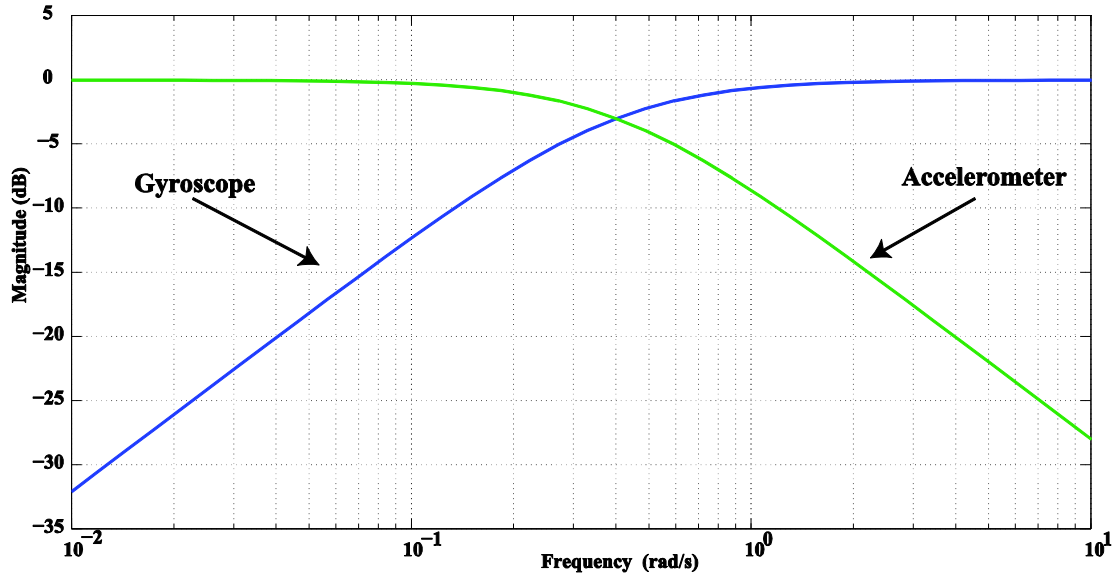


Figura 2.7 Respuesta en frecuencia del observador considerando la contribución del giroscopio y la contribución del acelerómetro

Es posible evaluar la capacidad de rechazo al ruido del ángulo de alabeo estimado por medio de un análisis a la respuesta en frecuencia del observador. En particular, la respuesta en frecuencia para el ángulo de alabeo estimado del acelerómetro y del giroscopio se muestra en la Figura 2.7. Esta figura muestra que el observador crea un filtro complementario de las estimaciones de alabeo del giroscopio y acelerómetro con un ancho de banda igual al polo del observador. Las estimaciones del acelerómetro son dominantes a bajas frecuencias mientras que las estimaciones del giroscopio son dominantes en altas frecuencias. Como resultado, ambas estimaciones son dominantes a las frecuencias donde cada sensor presenta menor ruido.

2.4 Diseño del controlador

En esta sección se diseña un controlador para el ángulo de alabeo del banco de pruebas con las especificaciones de la Tabla 2-2.

Tabla 2-2 Especificaciones de control

Característica	Especificación
Ancho de banda (Wb)	10-20 rad/s
Margen de fase	Superior de 50°
Margen de ganancia	Arriba de 12dB
Rechazo a perturbación de ruido del sensor	Arriba de 20dB para $\omega > 100$ rad/s
Rechazo de perturbación en la salida	Arriba de 20dB para $\omega < 1$ rad/s

El ancho de banda especificado fue seleccionado de acuerdo al tiempo de reacción en lazo abierto del banco de pruebas. Las otras especificaciones son valores típicos para el control de sistemas mecánicos.

Un control que cumple con los requerimientos es:

$$c_{\phi}(s) = \frac{(s+1.5)^2}{(s+0.07)(s+60)} \quad (14)$$

El controlador (14) fue diseñado utilizando las técnicas clásicas de Bode y está compuesto por un compensador de adelanto y un compensador de atraso. El polo más lento en $s=-0.07$ actúa de forma similar a un integrador, proporcionando una ganancia alta en lazo abierto a bajas frecuencias. El uso de una integral real fue evitado para reducir la manifestación del stick-slip, que normalmente es provocado por la interacción de la fricción estática con la alta ganancia que hay en las dinámicas lentas del controlador. La dinámica stick-slip es altamente no lineal y es difícil de modelar. Ésta introduce ciclos límite donde el estado de la posición está atorado mientras los otros estados (normalmente los estados del controlador) están en movimiento. La evolución del movimiento de los estados induce al estado de la posición a deslizarse repentinamente hacia una nueva

posición donde el ciclo comienza a repetirse por sí mismo. Stick-slip es en sí mismo un problema de control digno de un estudio por separado y no va a ser cubierto en esta tesis. Más información con respecto a este fenómeno se puede encontrar en [10], [11] y [12].

Considerando el controlador (14) y el modelo (9), la respuesta en frecuencia del sistema en lazo abierto se muestra en la Figura 2.8. Esta figura muestra que el ancho de banda y las especificaciones de robustez mostradas en la Tabla 2-2 se alcanzan utilizando el controlador (14). Por otro lado el análisis de la sensibilidad se presenta en la Figura 2.9.

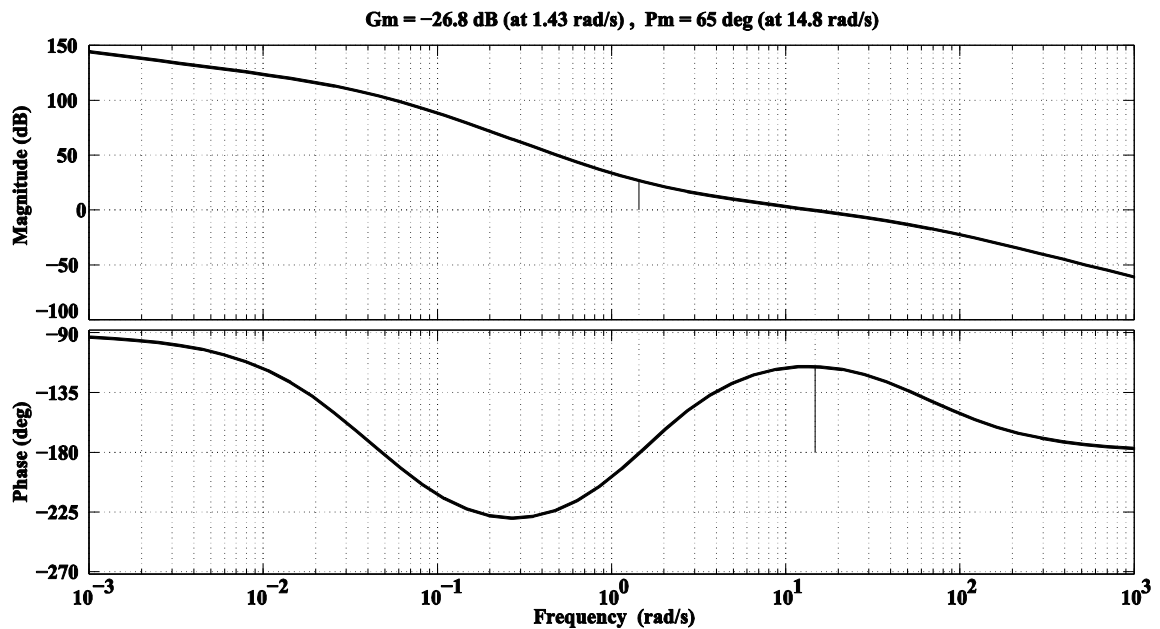


Figura 2.8 Respuesta en frecuencia de lazo abierto del ángulo de alabeo considerando el controlador (14)

La Figura 2.9 muestra que las especificaciones para el rechazo a perturbaciones de ruido del sensor y las perturbaciones a la salida se alcanzan con el controlador (14). No obstante, esta figura también muestra que para esta planta un alto grado de perturbación en la salida no es indicativo de un alto grado de rechazo a perturbaciones de entrada (IPR, input perturbation rejection, por sus siglas en ingles). Esto es debido a que (9) tiene una ganancia alta en lazo abierto a

frecuencias bajas. Esta es una característica muy importante de la planta porque las perturbaciones generadas por las ráfagas de viento normalmente son modeladas como perturbaciones de entrada en vez de perturbaciones de salida. Esto indica que un buen grado de IPR también es requerido para rechazar las perturbaciones debido a las ráfagas de viento. En este contexto, la Figura 2.9 muestra que el sistema tiene buen nivel de IPR para las frecuencias mayores al ancho de banda del sistema. Sin embargo, para el rango de frecuencias medias el nivel de IPR disminuye y para las frecuencias bajas se establece alrededor de -5dB. Aparentemente sería fácil lograr un grado mayor de IPR para frecuencias bajas si se reemplazara el polo en $s=-0.07$ de (14) con un integrador puro; sin embargo, esto magnificaría el movimiento del stick-slip. Para esta aplicación fue evitado y se prefirió un nivel bajo de IPR a bajas frecuencias.

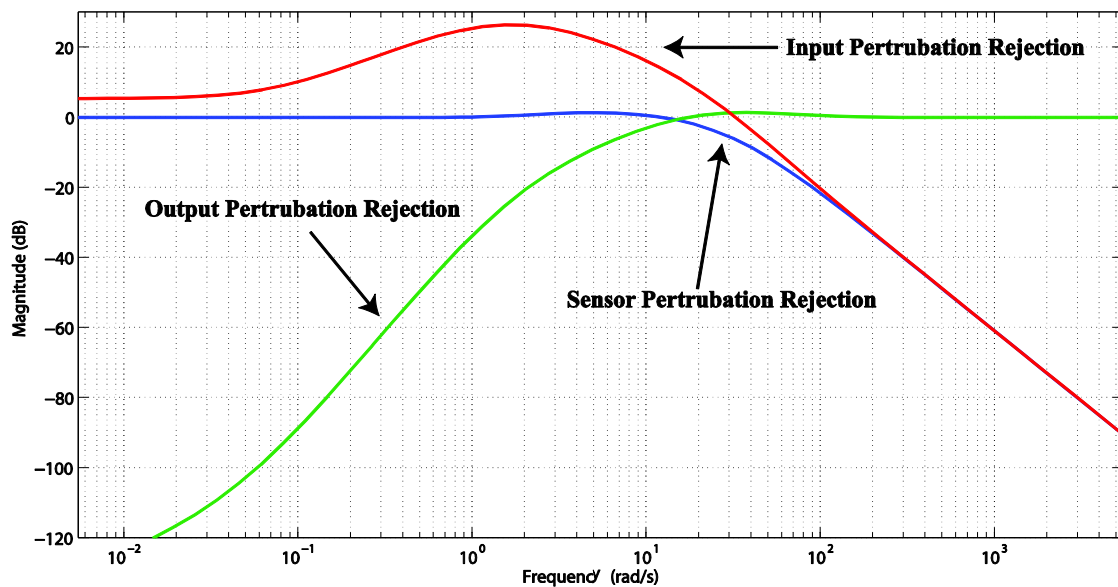


Figura 2.9 Características del rechazo a perturbaciones

2.5 Programación del Piccolo F28069 controlSTICK

Como se sabe el código se realiza en Simulink, sin embargo debido a la falta de bloques para mantener la comunicación por medio de I2C con los sensores, el uso del Stateflow y el generar la señal PWM se vuelve algo complicado.

La siguiente figura muestra el código general del PVTOL:

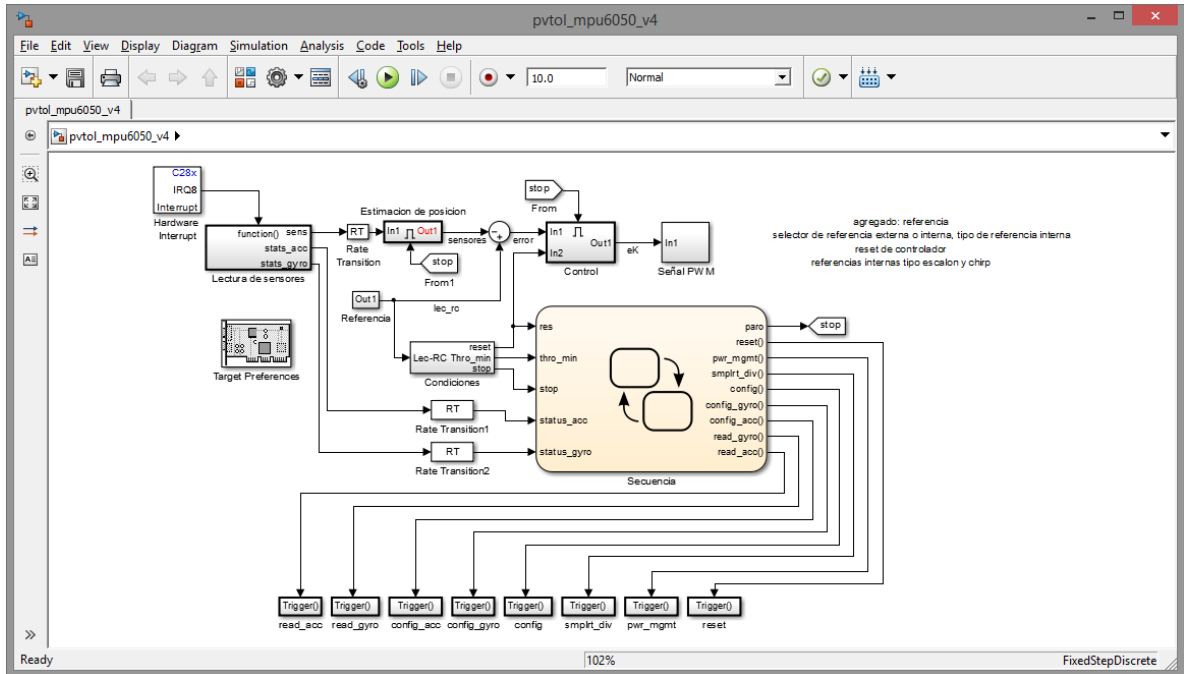


Figura 2.10 Código general del PVTOL

La explicación del código va a tener el siguiente orden:

- Primero se va a detallar todo lo relacionado con la comunicación por I2C que tiene el Piccolo con los sensores, esto incluye su configuración y la captura de datos.
- La segunda parte es sobre los algoritmos utilizados para la estimación de la posición angular de alabeo.
- La tercera etapa trata de la captura de la referencia externa y la generación de la referencia interna.
- Como cuarto paso está el controlador.
- Por último está la construcción de la señal PWM que va hacia los motores.

2.5.1 Comunicación I2C

Existen los bloques para transmitir y recibir datos por medio de I2C, pueden ser usados en cualquier modo de operación del Piccolo ya sea como maestro o como esclavo. Los bloques se muestran en la Figura 2.11.

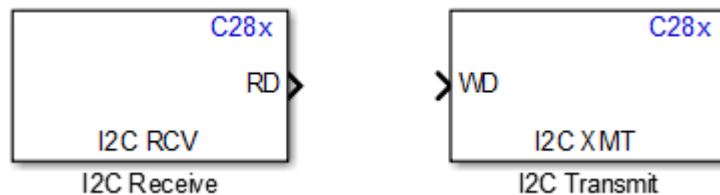


Figura 2.11 Bloques de recepción y transmisión por I2C

La dirección del dispositivo al que va a escribir se tiene que escribir en escala decimal. El bloque para transmitir sólo cuenta con una entrada para los datos a enviar, en una operación sencilla cumple con enviar sólo un dato. Sin embargo, si se requiere escribir en algún registro en específico hay que generar un vector en el cual primero se encuentre el registro seguido del dato a transmitir. Para lograr esto sólo se utiliza un *Mux* de dos entradas donde la primera es el registro y el segundo el dato a escribir. Como el bloque requiere que estén en escala decimal y la mayoría de las hojas de datos de los sensores utilizan la escala hexadecimal se recomienda utilizar el comando `hex2dec('XX')`, donde `XX` es cualquier número en hexadecimal, para hacer la conversión de hexadecimal a decimal de manera automática. La ventaja de utilizar esta función es que se muestre el número en hexadecimal y así en un futuro cualquiera pueda encontrar de manera sencilla cierto registro en la hoja de datos del sensor utilizado. Por último se activa la condición de *stop* para detener la comunicación.

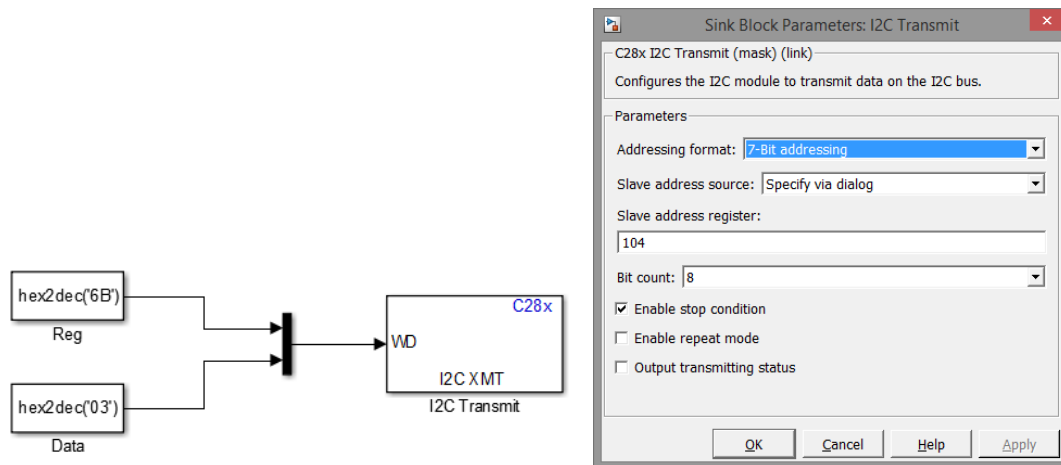


Figura 2.12 Configuración básica para transmitir un dato por medio de I2C

En la Figura 2.12 se muestra la configuración básica para escribir un número en un registro de un dispositivo esclavo conectado al protocolo I2C. Por ejemplo, envía el dato 03h al registro 6Bh del dispositivo esclavo con dirección 104d.

La transmisión de datos o escritura se ocupa para la configuración de los sensores y que tengan un funcionamiento correcto. Los registros que se configuran son cinco en el caso del sensor MPU6050 con dirección 68h o 104d:

6Bh – Power management 1.

En este registro se configura el modo de operación del sensor y la fuente de reloj. Al inicio se utiliza este registro para resetear al sensor y se vuelve a usar para definir la fuente de reloj, que siguiendo la recomendación de la hoja de datos se elige el eje z del giroscopio. Este registro se configura con el valor 08h para resetear al sensor y después con 03h para seleccionar la fuente de reloj.

19h – Sample Rate Divider

Este registro especifica la división de la salida del giroscopio para generar la velocidad de muestreo. Este registro se configura con 00h.

1Ah – Configuration

Este registro configura la sincronización externa de la estructura del muestreo de los pines y un filtro digital pasa bajas para el acelerómetro y el giroscopio. Este registro se configura con 00h.

1Bh – Gyroscope configuration

Este registro se utiliza para seleccionar la escala del rango de operación del giroscopio. Este registro se configura con 00h.

1Ch – Accelerometer configuration

Este registro sirve para configurar la escala del rango de operación del acelerómetro. Este registro se configura con 00h.

Cada registro tiene que estar dentro de un subsistema tipo *Function-call* debido a que se van a activar utilizando un *Stateflow Chart*, en la sección 2.5.2 se explica a detalle el uso del *Stateflow Chart*. Además se recomienda renombrar los subsistemas según el registro al que contenga dicho subsistema. Cuando se terminan de escribir en los registros anteriores se continúa con la lectura de las mediciones de los sensores.

El bloque para recibir datos tiene la peculiaridad de leer por defecto el registro cero de cualquier sensor o dispositivo esclavo y no cuenta con la opción para definir el registro en específico del cual se quiere tomar lectura. Por lo tanto, tomando en cuenta la estructura general del protocolo I2C para leer datos de cualquier registro, se realizan los siguientes pasos:

- 1) El Piccolo cuenta con siete interrupciones básicas dentro del grupo PIE 8. La interrupción que se utiliza es ARDY (Register-access ready condition), cuando se activa esta interrupción se indica que es posible acceder a los registros. Para poder hacer uso de ella es necesario activarla en el bloque *Target Preferences* en la pestaña *Peripherals* y en el apartado de I2C se activan las casillas *Enable system interrupt* y *Enable ARDY interrupt* como lo muestra la Figura 2.13.

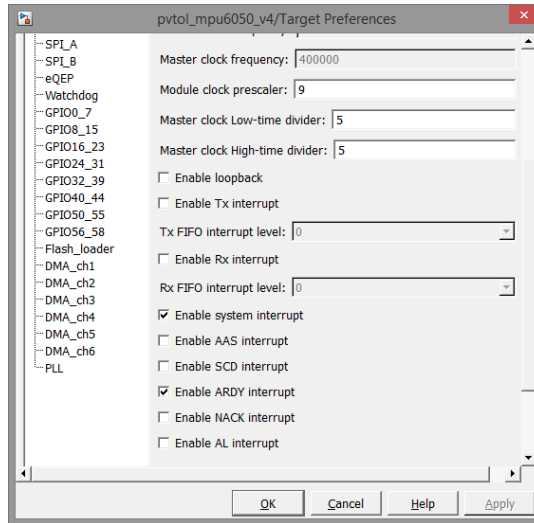


Figura 2.13 Configuración para activar la interrupción

- 2) Se agrega un bloque de transmisión sin activar la opción de la condición de stop, cuya función es escribir el registro deseado.

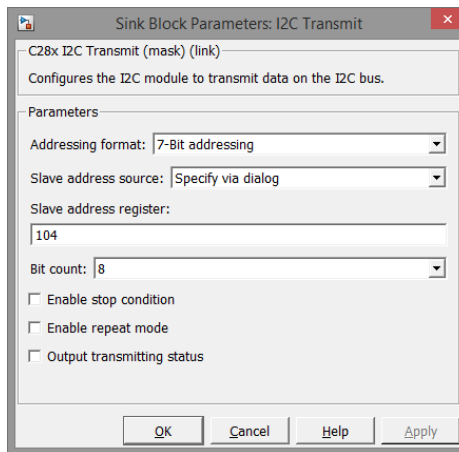


Figura 2.14 Primera parte de la comunicación para leer cualquier registro por I2C

Cada sensor tiene su apartado de registros en donde se almacenan las mediciones, por lo tanto se requieren dos transmisores I2C con el registro

correspondiente a cada sensor. En el caso de giroscopio el registro es el 43h mientras que para el acelerómetro es el 3Dh.

- 3) Generar la rutina que se va a ejecutar cuando se activa la interrupción. En este caso se necesita el bloque *Hardware Interrupt* y se configura según las interrupciones a utilizar, es decir, definir que elemento va a generar la interrupción; como es una interrupción de I2C, ésta se almacena en INT8.1 (según la tabla 1-119 de [13]) por lo tanto el bloque se configura de la siguiente forma: el número de la interrupción del CPU es 8, para la interrupción PIE 1, la prioridad de *Simulink* es 12, este valor se obtiene de la tabla 1-120 de [13], y al final se escribe 0 y el resultado se muestra en la Figura 2.15:

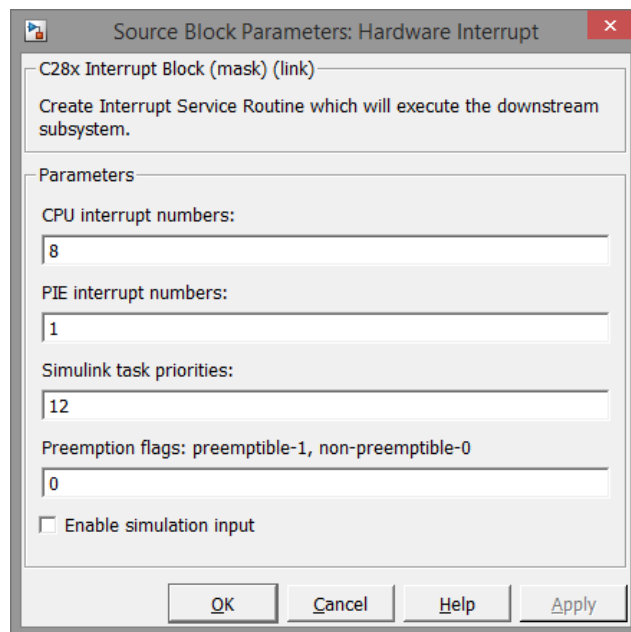


Figura 2.15 Configuración del bloque *Hardware Interrupt*

El bloque *Hardware Interrupt* se conecta a un subsistema tipo *Function-call* que se activa luego de ser activada la interrupción, por lo cual dentro de este subsistema esta la segunda parte de la comunicación que es recibir o leer datos por I2C.

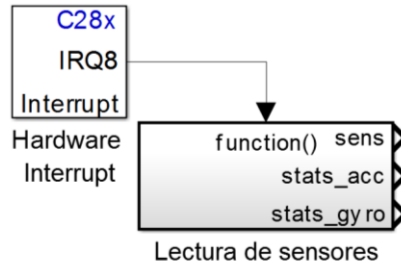


Figura 2.16 Subsistema para ejecutar código con interrupción

- 4) Monitorear el registro I2CDXR (I2C data transmit register). Este registro almacena el dato a transmitir por I2C, por eso es que se monitorea para determinar el sensor que se va a leer. Dentro del subsistema creado en el paso anterior se agrega el bloque Memory Copy y se configura como lo muestra la Figura 2.17.

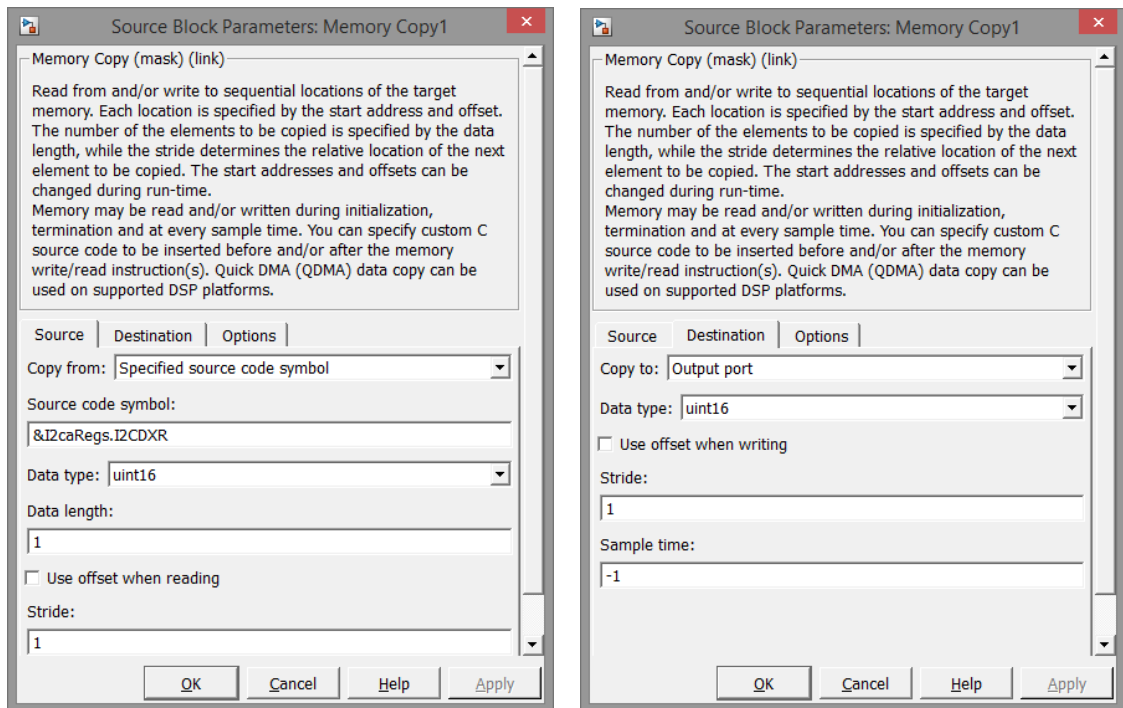


Figura 2.17 Configuración del bloque Memory Copy

Se utilizan comparadores de constantes para determinar el registro a leer. Cada comparación tiene la condición "igual a" y comparan el dato

monitoreado con la constante que es el registro correspondiente a los sensores. Para el giroscopio se compara con 67d mientras que para el acelerómetro es 61d. Estos son los mismos registros que se definen en el paso 2 sólo que en escala decimal ya que el comparador necesita que la constante este en escala decimal.

Cuando se cumple la condición el comparador genera una señal booleana 1, mientras que 0 si no se cumple la condición. Tomando en cuenta la salida generada se incluyen dos subsistemas con una condición tipo *Enable* uno para cada sensor. De esta manera, cuando la señal de algún comparador sea 1 se activa el subsistema correspondiente y se ejecutan los algoritmos que están dentro. El código se muestra en Figura 2.18.

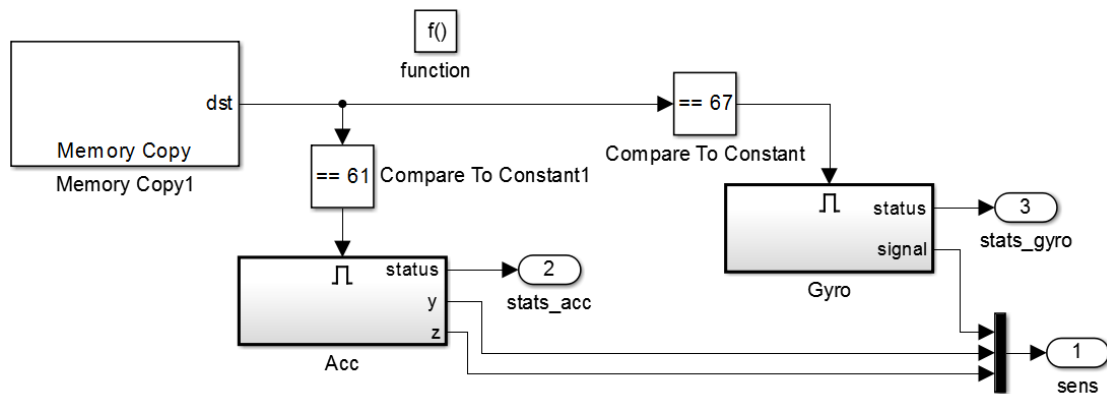


Figura 2.18 Código para determinar el sensor a leer

- 5) Lectura y concatenamiento de bytes. Para la lectura de cada sensor se necesita de un bloque I2C Receive en cada apartado, es decir, uno para el acelerómetro y otro para el giroscopio. Las configuraciones que se necesitan hacer en cada bloque son: definir la dirección del sensor en escala decimal que es 104d, activar la condición de stop para cerrar la comunicación I2C con el sensor, activar la salida de la bandera estatus (ya que es útil en el uso de Stateflow) y el tiempo de muestreo -1 como lo muestra la Figura 2.19. Estas configuraciones son iguales para el giroscopio y acelerómetro a excepción de la cantidad de registros a leer. En el caso del giroscopio sólo se necesita tomar lectura del eje “Y” por lo

cual el tamaño del dato a leer es 2, mientras que para el acelerómetro el tamaño a los datos a leer es 4 porque se ocupan los ejes “Y” y “Z”.

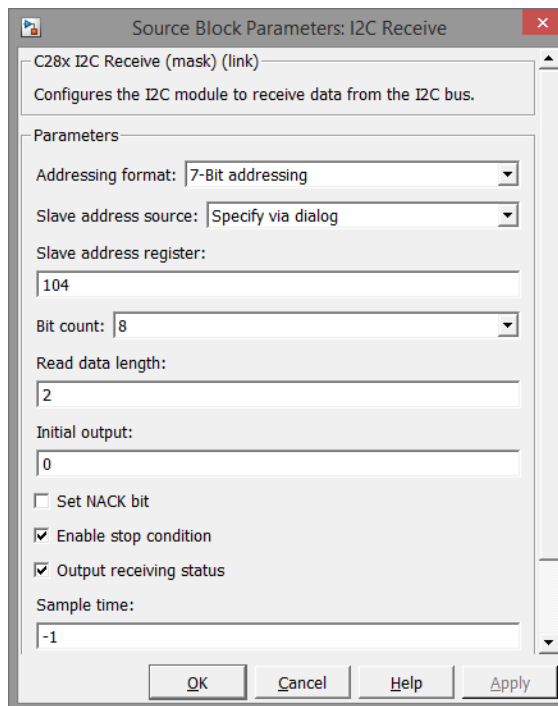
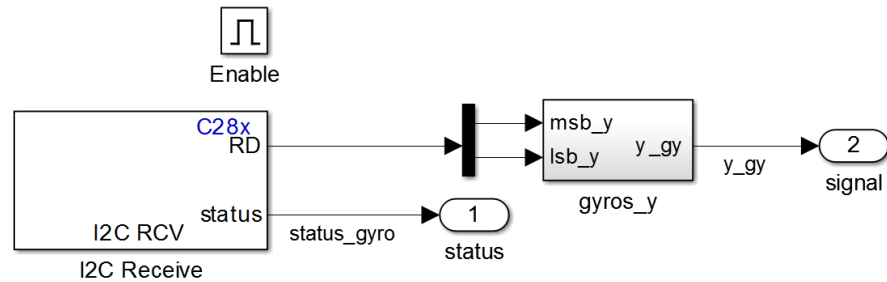


Figura 2.19 Código y configuración para leer datos por I2C

Como el tamaño de la medición del eje “Y” es de 16 bits, el sensor lo divide en 2 partes de 8 bits debido a que las memorias son ese tamaño, por lo que es necesario concatenar las dos partes correspondientes a un eje y así formar la medición completa.

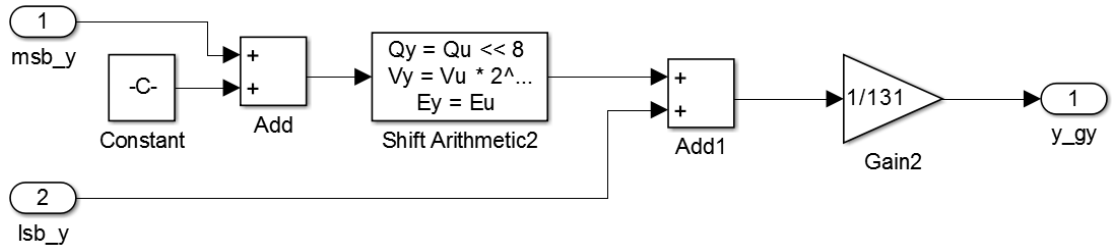


Figura 2.20 Código para concatenar la medición

Tomando como ejemplo los dos números que recibe el Piccolo del giroscopio, el primero corresponde a los 8 bits más significativos MSB y el segundo número incluye los 8 bits menos significativos LSB.

Por ejemplo, si MSB = 00100111 y LSB = 11001101; la secuencia para unir ambas partes es la siguiente:

- a) El número MSB se suma con una constante cero de 16 bits y el resultado de la suma es el MSB pero de 16 bits.

$$\text{MSB} + 0000000000000000 = 000000000100111$$

- b) Se realiza un corrimiento al MSB de 8 bits hacia la izquierda.

$$\text{MSB} \ll 8 = 0010011100000000$$

- c) El MSB resultante del corrimiento se suma con el LSB.

$$\text{MSB} + \text{LSB} = 0010011111001101$$

- d) Al final se cambian las unidades de la lectura, en el caso del giroscopio el resultado se divide entre 131 para que las unidades estén en °/seg, mientras que para las mediciones del acelerómetro se divide entre 16384 para que las unidades estén en g (gravedad, $1g = 9.81m/s^2$).

- 6) Leer estatus. Cuando la comunicación termina se activa la bandera del estatus.

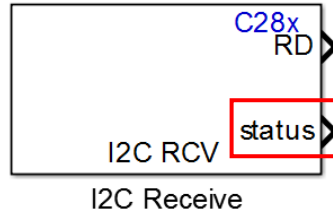


Figura 2.21 Bloque para leer por I2C con la bandera estatus

2.5.2 Stateflow chart

Como una máquina de estados el Stateflow Chart determina el orden de los registros a configurar en el sensor y después el orden de la captura de datos. Cuenta con funciones de estado y funciones de transición. Las funciones de estado pueden funcionar de tres formas distintas, al entrar en el estado, durante el estado y cuando sale del estado. Las funciones de transición son usadas principalmente para cambiar de un estado a otro.

El funcionamiento que realiza este bloque es: en cada estado se activa una función que indica el registro al que se va a escribir y la transición de un estado a otro se realiza hasta que termina de ejecutarse el estado anterior. Después de pasar por los estados correspondientes a las configuraciones, comienza un lazo en el cual toma lectura del giroscopio, cuando termina la medición se lee el estatus generado y se toma como condición para avanzar al siguiente estado que es la medición del acelerómetro y siguiendo el mismo proceso de leer el estatus y tomarlo como condición para que se repita la medición del giroscopio. Aquí permanece el sistema hasta que algún interruptor se active y el sistema deja de tomar mediciones de los sensores, además se detiene el controlador para detener los motores.

Los pasos para generar el diagrama de flujo de los estados del Stateflow es el siguiente:

- 1) Agregar el bloque *Chart (MATLAB)* que se encuentra en la librería de Stateflow.

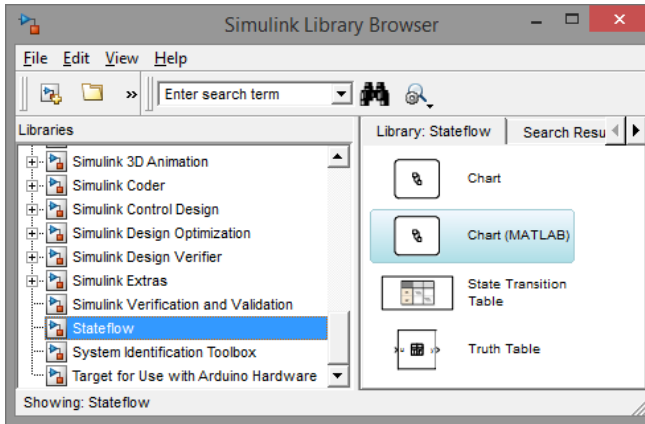


Figura 2.22 Librería de Stateflow

- 2) Dentro del bloque Chart hay que agregar los estados. El icono está en el lado izquierdo y tiene la forma de un rectángulo. Para cada estado se recomienda numerar y escribir el nombre de la función que va a activar con punto y coma al final. Para mantener un orden, cada función lleva el nombre del registro a activar según la hoja de datos del sensor. De esta manera es más simple rastrear que registro es y qué características tiene dicho registro.
- 3) Definir las funciones de salida. Estas funciones se definen desde el *Model Explorer* que se encuentra en la barra de herramientas. Para agregar una función se utiliza el icono en forma de rayo naranja, al seleccionarlo aparece con el nombre "event", después se renombra según el registro que se desee y en el apartado de Scope se selecciona como salida hacia Simulink y guardar los cambios, se tiene que hacer para todas las funciones correspondientes a registros.
En el caso de agregar las variables que serán de entrada y salida, se agregan con el icono a la izquierda del usado para las funciones. El proceso es similar se tienen que renombrar y definir si son entradas o

salidas. La Figura 2.23 muestra todas las funciones y variables que se definen.

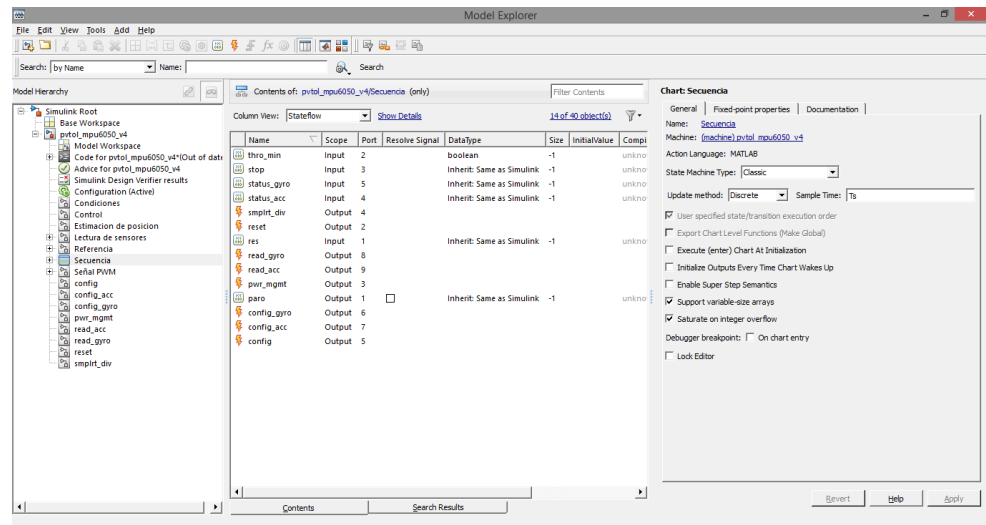


Figura 2.23 Definición de variables y funciones de Stateflow

- 4) Construir la secuencia de los estados. Al primer estado se le conecta una transición por defecto, para agregar esta transición es seleccionando el icono que es una flecha apuntando hacia abajo y a la derecha. Esta transición indica cual estado va a iniciar primero, en este caso es el reset, después para conectar los estados lo que se hace es acercar el puntero del mouse a una cara del estado hasta que el puntero cambie de forma a una cruz negra, pulsar, mantener y arrastrar al estado que se quiere conectar, al hacer esto se genera una flecha con la línea punteada de color rojo y soltar cuando la flecha cambie a color azul y con una línea continua.

Para generar o definir las funciones de transición se selecciona la transición hasta que aparezca un signo de interrogación dentro de un cuadro de texto, se borra el signo y se escribe la función de transición. La primera función es un retardo de 5 segundos para permitir que los ESC se inicialicen correctamente, esta función es $after(t,sec)$ donde t es el tiempo que se tiene que esperar para cambiar de estado. Las

siguientes funciones de transición se definen utilizando las variables de entrada y comparándolas para que sean igual a unas constantes, el símbolo es == para las variables *res*, *stop* y *thro_min*, las cuales se comparan con 1, mientras que para las variables *status_acc* y *status_gyro* la constante de comparación es 13360, debido que es lo que se genera al terminar de recibir información del sensor. El flujo de los estados queda como la Figura 2.24.

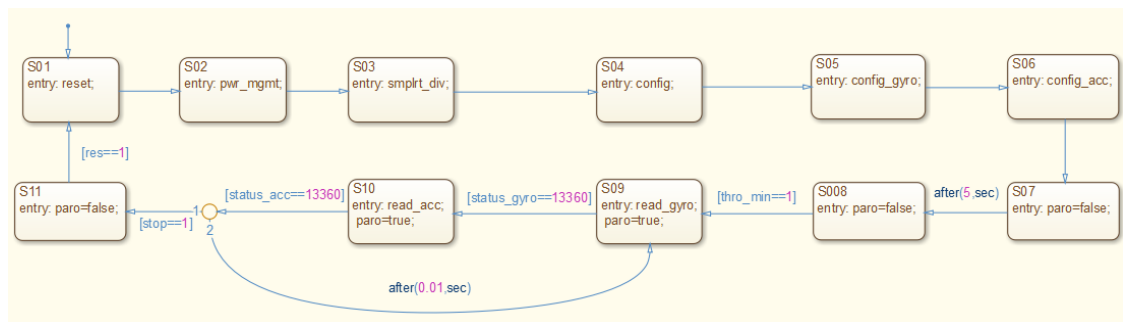


Figura 2.24 Flujo de los estados del Stateflow

Tabla 2-3 Descripción de los estados del Stateflow

Estado	Función
S01	Reset del sensor MPU6050
S02	Selecciona la fuente del reloj del sensor
S03	Define la velocidad de muestreo del giroscopio
S04	Configura un filtro digital paso bajo para el acelerómetro y giroscopio
S05	Selecciona el rango de operación del giroscopio, en este caso es ± 250 °/s
S06	Selecciona el rango de operación del acelerómetro, en este caso es $\pm 2g$
S07	Define la condición del paro en 0, tiene un retardo de 5 segundos para cambiar al siguiente estado.

S08	<p>Mantiene la condición del paro en 0, no cambia de estado hasta que la condición thro_min sea igual a 1.</p> <p>Esta condición es para asegurar que el radiocontrol este en la posición cero.</p>
S09	<p>Inicia el proceso para la lectura del eje Y del giroscopio.</p> <p>No cambia de estado hasta que se cumpla la condición status_gyro == 13360, la variable que compara es el estatus generado al terminar la lectura del giroscopio.</p> <p>Define la condición de paro en 1 para activar el cálculo de los algoritmos de estimación de posición, control y además permite la salida de la señal PWM a los motores.</p>
S10	<p>Inicia el proceso para la lectura de los ejes Y y Z del acelerómetro.</p> <p>No cambia de estado hasta que se cumpla la condición status_acc == 13360, la variable que compara es el estatus generado al terminar la lectura del acelerómetro.</p> <p>Mantiene la condición de paro en 1.</p>
S11	<p>Para llegar a este estado se tiene que activar cualquier interruptor.</p> <p>Escribe 0 en la salida paro lo que significa que se desactivan las salidas a los motores y los cálculos de los algoritmos de control y estimación de posición.</p> <p>Para cambiar al estado S01 se tienen que activar los dos interruptores de manera simultánea.</p>

- 5) Conectar las salidas de las funciones a los subsistemas correspondientes. Por último sólo se conectan las salidas del Stateflow a los subsistemas creados con los registros a escribir al sensor. El resultado se muestra en la Figura 2.25.

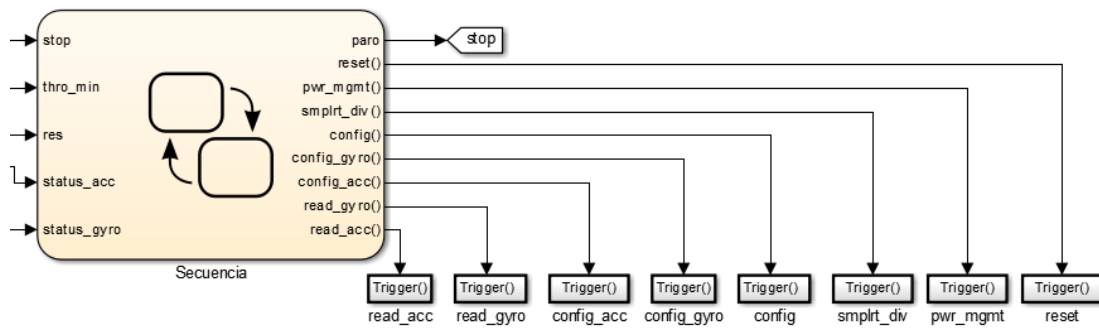


Figura 2.25 Bloque Stateflow con las funciones de salida

2.5.3 Condiciones de entrada

El Stateflow necesita cinco condiciones para avanzar de cierto estado al siguiente. Dos de estas condiciones son los estatus que generan los bloques Receive I2C que indican cuando ha terminado de recibir información del sensor, por lo que estas condiciones se conectan directamente de la fuente que las genera. Sin embargo las otras tres condiciones se utilizan para:

- asegurarse que la entrada de referencia sea cero
- detener el sistema si se activa un interruptor
- reiniciar la secuencia del Stateflow.

En el primer caso se toma la señal de la referencia y se compara con la condición mayor e igual que (\geq) el valor mínimo que se genera al tener el canal de radiocontrol en la posición cero. Cuando se cumple la condición el comparador genera una salida booleana igual a uno.

Para los otros dos casos que utilizan los interruptores se generan de la siguiente manera. Primero se definen los pines de entrada en donde se conectan los interruptores al Piccolo con el bloque *Digital Input*, en este bloque se selecciona el pin y que tipo de señal recibe. Como son dos interruptores se necesitan dos bloques, cada uno con su pin definido y con tipo de señal a recibir booleana. Los pines son: *GPIO6* y *GPIO4*.

Para detener el sistema en caso de que se active cualquier interruptor se utiliza la compuerta lógica OR con la señal de salida tipo booleana. Mientras que para reiniciar el sistema se usa la compuerta lógica AND porque para el reinicio se tienen que presionar ambos interruptores.

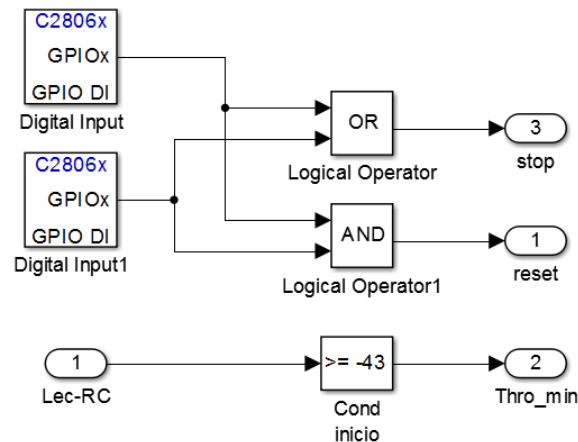


Figura 2.26 Condiciones de entrada al Stateflow

2.5.4 Estimación de la posición angular

El cálculo de la posición angular se puede hacer con el acelerómetro utilizando solamente los ejes Y y Z con la siguiente operación $\arctan(y/z)$, el resultado es en radianes por lo que se necesita hacer una conversión a grados. Sin embargo al intentar medir algún movimiento de altas frecuencias las mediciones no son correctas.

También se puede obtener la posición angular a partir del giroscopio ya que este censa la velocidad angular, por lo cual con tan sólo integrar esta medición se obtiene la posición angular. A diferencia del acelerómetro responde bien a altas frecuencias pero presenta fallas en bajas frecuencias y además al hacer el proceso de integración la señal resultante presenta deriva.

Como una posible solución se plantea estimar la posición angular con un observador de Luenberger. Este observador cumple la función de un filtro, está diseñado para observar a un integrador ya que el giroscopio mide la velocidad angular y con el acelerómetro se calcula una posición angular, estas señales son

las entradas del observador y donde la salida es la posición angular estimada a partir de ambos sensores. Con esto cuando el sistema se comporte a bajas frecuencias la señal dominante es del acelerómetro mientras que a medida que la frecuencia vaya aumentando la señal dominante sea la del giroscopio.

Como un pre-filtro para mejorar las señales se incluye una banda muerta al giroscopio de -3 a 3.

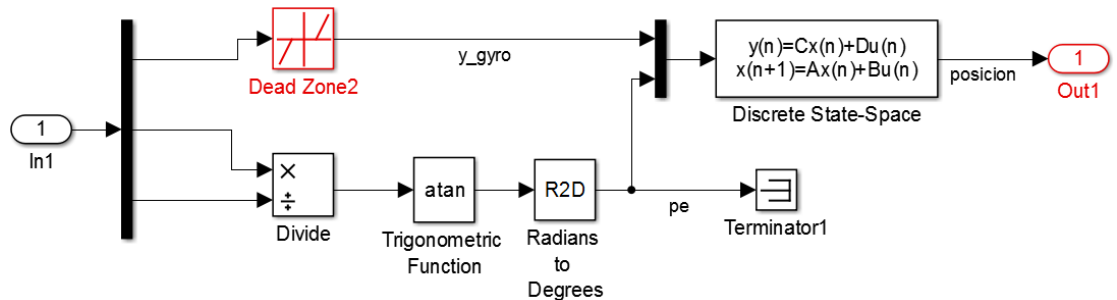


Figura 2.27 Código para estimar la posición angular de alabeo

2.5.5 Referencias del sistema

Como se mencionó en la sección 2.2.1, el sistema es capaz de recibir una referencia externa o generar dos tipos de referencias internas.

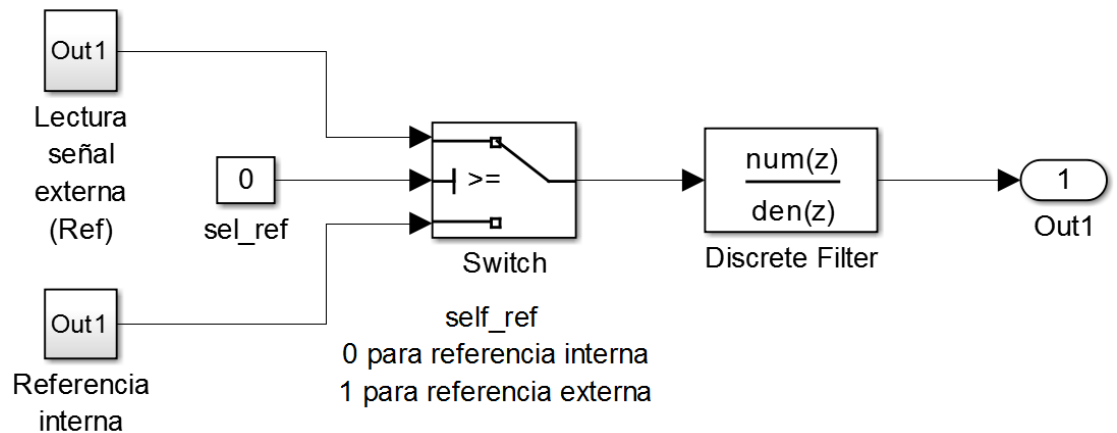


Figura 2.28 Código para seleccionar el tipo de referencia

La referencia externa se modifica con un radiocontrol, que para efectos prácticos se utiliza el canal 3 porque es el que puede mantener fija la posición deseada ya que los demás canales regresan a una posición central.

Para medir esta señal de entrada se ocupa el bloque *eCAP (Enhanced Capture Module)* que es un módulo dedicado a la medición del periodo y del ciclo de trabajo de una señal tipo tren de pulsos. Este módulo cuenta con un pin dedicado a la captura de señales con cuatro contadores de 32 bits capaces de detectar bordes positivos o negativos.

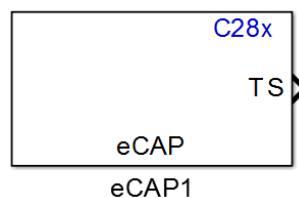


Figura 2.29 Bloque eCAP para capturar la señal del radiocontrol

Los contadores se configuran de tal manera que el primer contador comienza al detectar el cambio del pulso de 0 a 1 mientras que el segundo contador cuando detecta el cambio de 1 a 0, la diferencia de ambos contadores da como resultado la señal recibida del radiocontrol (contador 2 – contador 1).

Este modo de operación puede trabajar capturando una señal o generando una señal PWM, el modo de operación se selecciona en la primera pestaña.

En la pestaña eCAP se encuentran todas las configuraciones para la captura del tren de pulsos. La primera opción es un pre-escalamiento de la señal muy útil para señales de alta frecuencia. El modo de control puede ser continuo o de una sola captura, en este caso se selecciona el modo continuo para que siempre este recibiendo la señal del radiocontrol.

En el caso de la configuración de los eventos para la captura de la señal se basa en el ejemplo 2 (página 442 de [13]). Este ejemplo utiliza los cuatro eventos, con los primeros dos se obtiene el ciclo de trabajo positivo de la señal, para calcular

el periodo de la señal se ocupan los eventos 1 y 3. Estos eventos se configuran para los diferentes bordes del pulso, los eventos 1 y 3 se configuran para el borde positivo mientras que los eventos 2 y 4 utilizan el borde negativo. Aunque en este caso no se utilizan los eventos 3 y 4 se dejan activados para que en caso de ser necesario se obtenga el periodo de la señal. En los eventos 1 y 2 se activan sus respectivos reset de contador porque al realizar pruebas se detectó un pequeño error en la señal capturada, estos errores se generan al ejecutarse el reset de los eventos lo que ocasiona pulsos en la señal capturada.

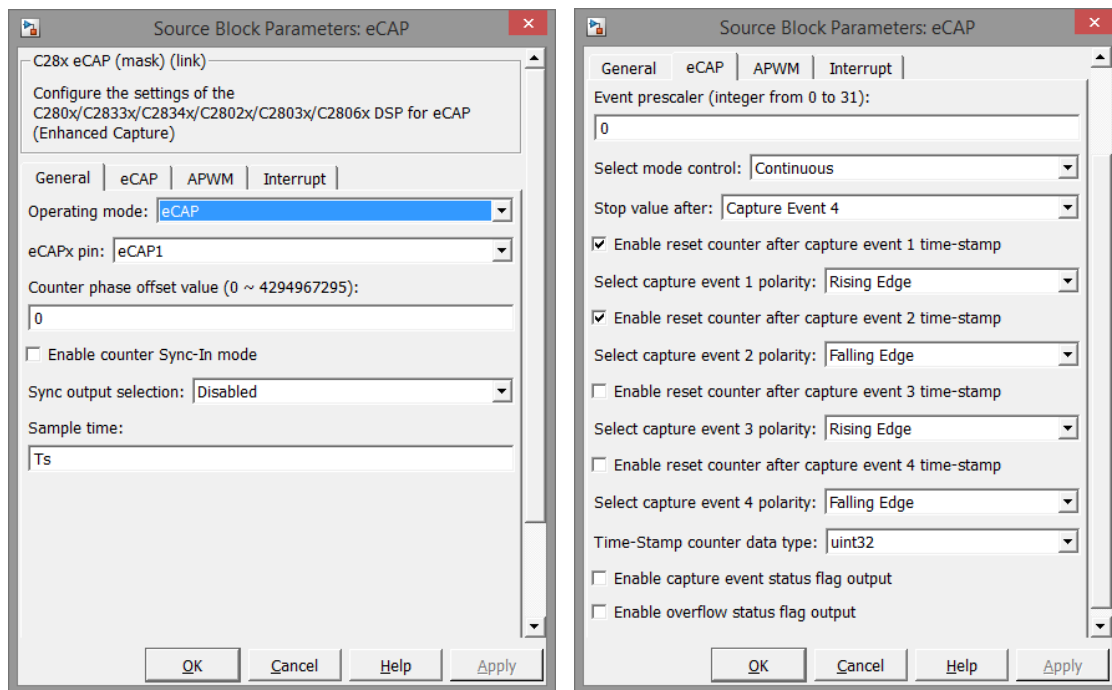


Figura 2.30 Configuración del bloque eCAP

El siguiente paso es escalar la señal recibida para que sea equivalente a grados por medio de las siguientes ecuaciones:

$$(l_{mia})(K_{S1}) + K_{S2} = smia$$

$$(l_{mxa})(K_{S1}) + K_{S2} = smxa$$

Donde l_{mia} es el valor mínimo capturado del radiocontrol, l_{mxa} el valor máximo capturado del radiocontrol, $smia$ la posición angular mínima en grados de

referencia, $smxa$ es la posición angular máxima en grados permitida al sistema, Ks_1 y Ks_2 son los coeficientes a obtener para que se realice el escalamiento.

El código resultante es el siguiente:

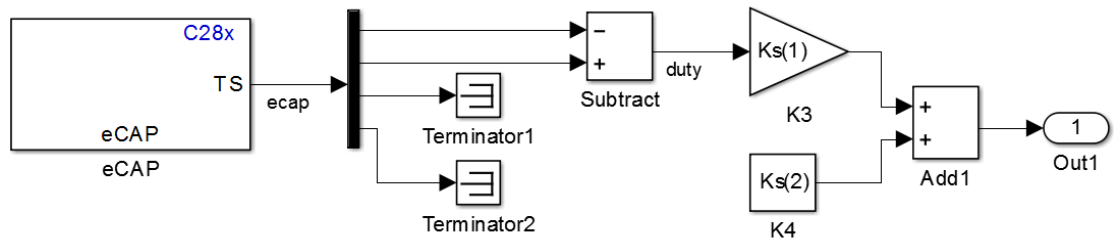


Figura 2.31 Código de adquisición de la señal del radiocontrol

Los números mínimo y máximo que captura del radiocontrol se obtienen monitoreando la variable *duty* en el CCS, este proceso de monitoreo se explica en la sección 2.5.9. Las posiciones angulares que se definen son -25 y 25 porque las posiciones angulares mínima y máxima que permite la estructura son -30 y 30 grados. Esta restricción se debe a los topes en donde están los interruptores de seguridad que al activarse detienen todo el sistema.

Las referencias internas son de dos tipos:

- La primera es una serie de escalones generada con el bloque *Repeating Sequence Stair* con el siguiente orden: 0, 8, 0, -8, y se repite indefinidamente.
- La segunda referencia es una señal tipo Chirp de 0.05Hz a 1Hz con una amplitud de 16 desde -8 hasta 8.

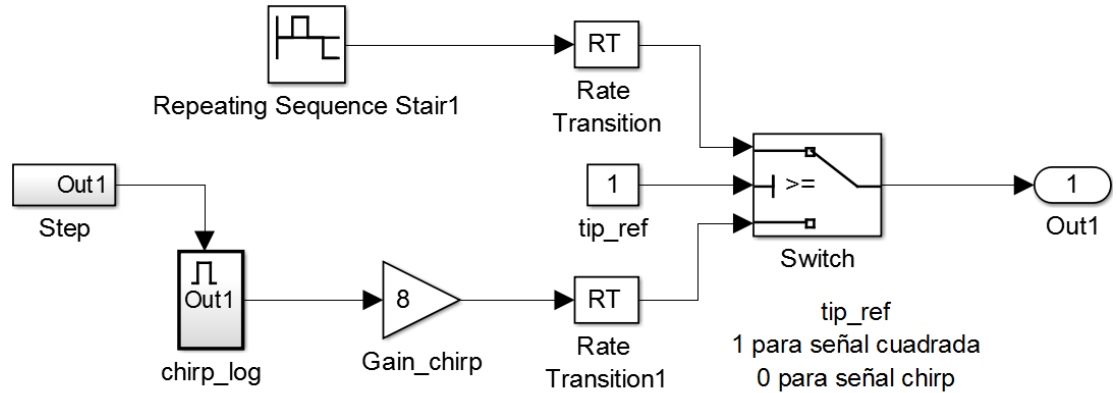


Figura 2.32 Código para seleccionar el tipo de referencia interna

Tanto para la selección del tipo de referencia interna como para la selección entre la referencia externa e interna se utiliza el mismo principio o estructura de programación. Esta estructura consta de dos bloques un *Switch* y una constante. Al bloque del número constante se renombra y se define como un número booleano el cual su función es ser la condición para el cambio de señal. La condición del *Switch* se define como $u2 \geq 0$ donde $u2$ es el puerto 2 de dicho bloque, por lo tanto configurando de esta manera al *Switch* y con un número booleano, cuando la constante sea 1 la señal que el *Switch* deja pasar es la que está conectada en el puerto 1 mientras que cuando sea 0 la señal del puerto 3 es la que pasa por el *Switch*. El código resultante se muestra en la Figura 2.32.

2.5.6 Controlador

Para manipular al controlador de forma sencilla se diseña como función de transferencia discreta con la ganancia separada para poder modificarla en caso de ser necesario. Esta sección del código funciona hasta que se comienza a capturar las mediciones de los sensores. También cuenta con una pequeña corrección a la salida del controlador para mejorar la respuesta es caso de que exista desbalance en los motores.

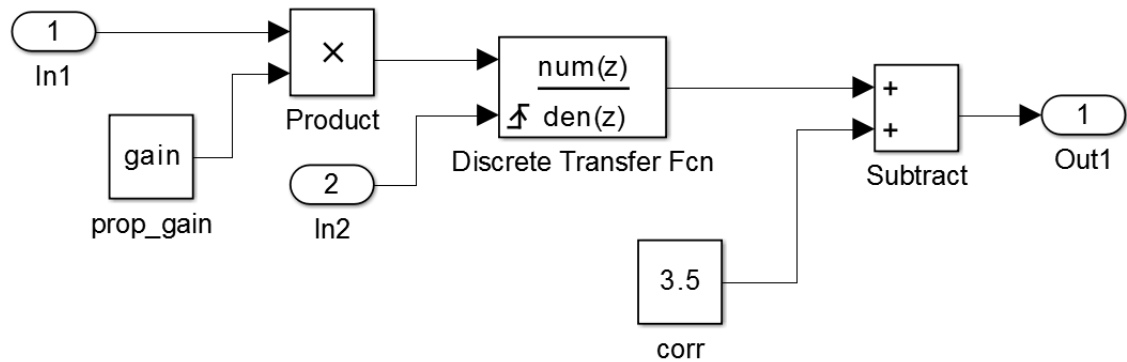


Figura 2.33 Código para el controlador

2.5.7 Modulación por ancho de pulso PWM

Los motores se controlan por medio de una señal PWM (Pulse Width Modulation), modulación de ancho del pulso. Como se tienen más salidas que entradas se plantea un arreglo en el cual con una sola entrada se controlen ambos motores. Este arreglo permite que funcionen de la siguiente manera:

- Si la señal de entrada es 0 los motores funcionan a la mitad de su capacidad lo que genera que el sistema esté en equilibrio
- Cuando la entrada sea 1 el motor1 acelera y el motor2 desacelera
- Cuando la entrada sea -1 el motor1 desacelera mientras que el motor2 acelera.

Después de este pequeño arreglo es necesario una saturación para evitar que la señal que se genere sobrepase los límites soportados por los motores y así evitar un mal funcionamiento e incluso que se dañen.

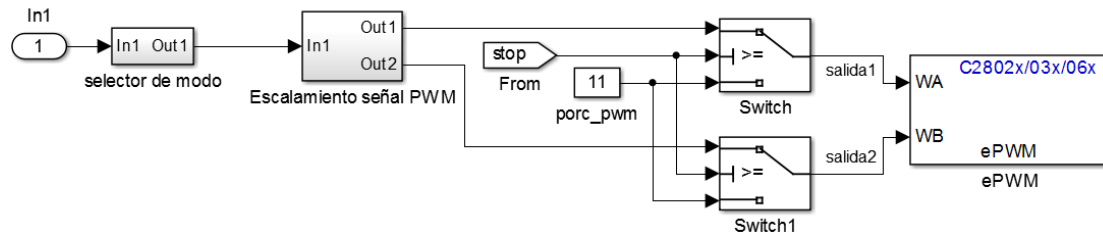


Figura 2.34 Código para generar las señales PWM

Para generar la señal PWM se utiliza el bloque *ePWM*, cada bloque sirve para generar dos señales. El *Piccolo* cuenta con ocho salidas para generar esta señal. El bloque cuenta con diferentes pestañas de configuración pero sólo se modifican tres. En la primera que es la configuración general se define que par de salidas de van a utilizar, en la versión *controlSTICK* del *Piccolo* sólo están disponibles los primeros cuatro, en las unidades del reloj para el periodo se seleccionan los ciclos de reloj. Como el periodo típico del PWM es de 20ms y las unidades están en ciclos de reloj se calcula el periodo en ciclos de reloj para que el periodo generado sea de 20ms, el resultado es 6250 ciclos de reloj. El contador que se escoge en esta aplicación es el contador Up-Down y se define el valor a pre-escalar la señal de reloj como 128. La Figura 2.35 muestra la configuración mencionada.

Cada salida PWM con la que cuenta el bloque opera de manera independiente una de la otra. Estas cuentan con cuatro eventos que son:

- PRD. Valor del periodo
- Zero. Contador igual a cero
- CMPA. Contador A
- CMPB. Contador B

Estos eventos se definen para realizar una acción que se refleja en la salida y que en conjunto generan la señal PWM. Las acciones posibles para definir los eventos son:

- Set High. Establece la salida en nivel alto.

- Clear Low. Establece la salida en nivel bajo.
- Toggle. Si la salida se encuentra en un nivel alto, la cambia a un nivel bajo. Mientras que si está en un nivel bajo, cambia a nivel alto.
- Do Nothing. Mantiene la salida en el mismo nivel lógico en el que está.

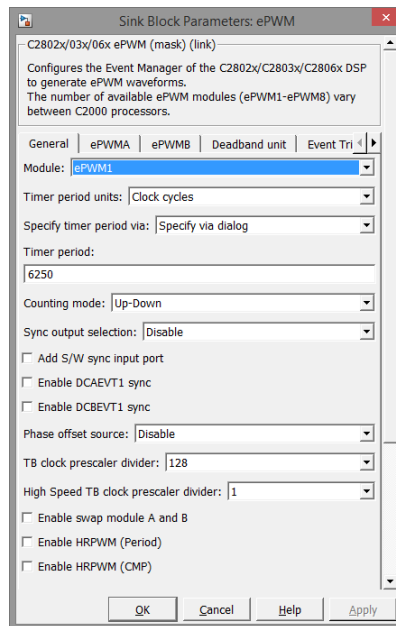


Figura 2.35 Configuración general del bloque ePWM

La siguiente configuración es la pestaña *ePWMA* que es donde se definen las acciones de cada evento para generar una señal PWM que va a manipular el primer motor, su configuración queda de la siguiente manera: se habilita esta salida, las unidades que se seleccionan para el *CMFA* son en porcentaje y como valor inicial cero. Por último se definen las funciones que tomaran los contadores disponibles en este pin, el contador *ZERO* se define en *Set* y el *CAU* como *Clear*, el resto se mantienen en *Do nothing*.

Para generar la otra señal PWM para el segundo motor la configuración de la pestaña *ePWMB* es igual a la pestaña anterior pero con diferencias en los eventos de los contadores que usa. En este caso el contador *ZERO* se define como *Set* y el contador *CBU* como *Clear* el resto se define como *Do nothing*.

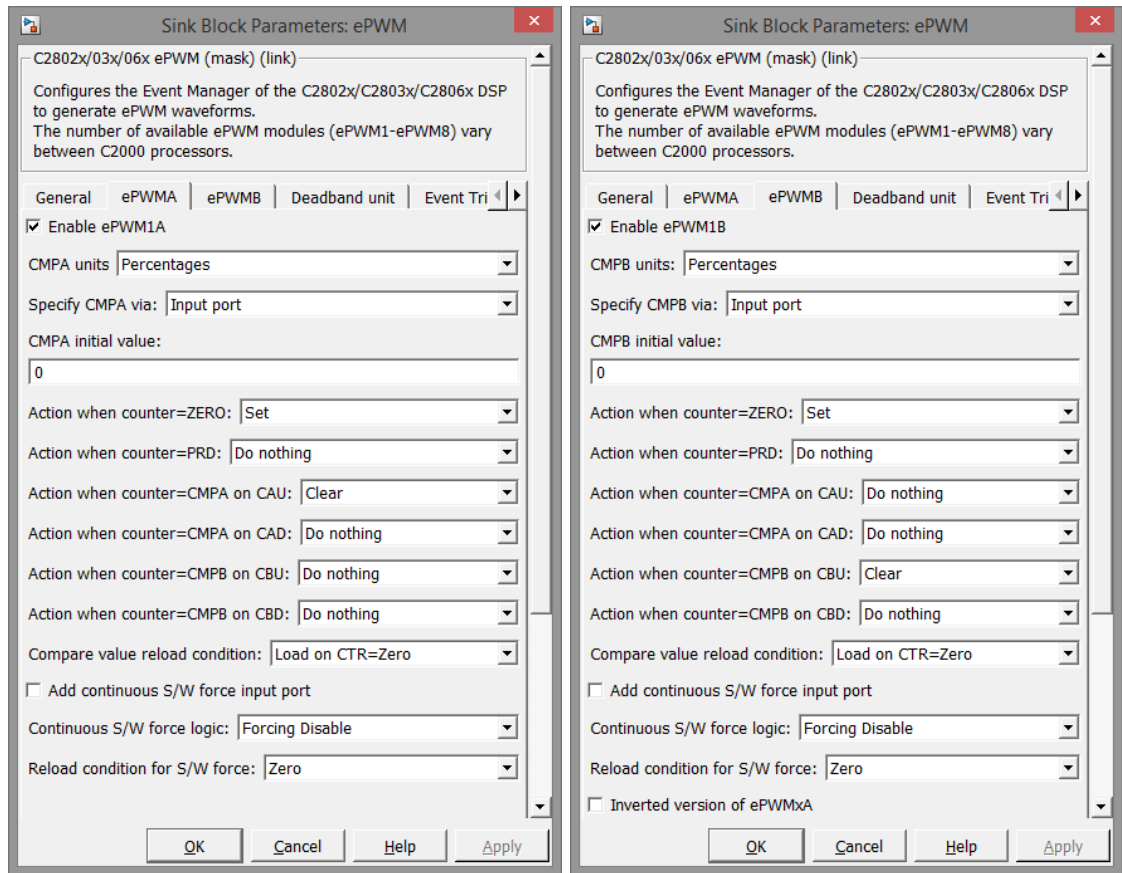


Figura 2.36 Configuraciones de las pestañas ePWMA y ePWMB del bloque ePWM

2.5.8 Script Matlab

En un script de matlab se definen las variables a utilizar, los algoritmos para la escalacion de las diferentes señales, las ecuaciones para el diseño del observador y controlador. De esta forma cuando se requiera modificar el controlador, tiempo de muestreo o cualquier variable del programa es muy sencillo hacerlo.

```

%pvto1
%variables del micro
Ts=0.001;%tiempo de muestreo
%escalamiento del acelerometro al primer motor
lmina=-25;
lmaxa=25;
smina=11;
smaxa=19;
a=[lmina 1;lmaxa 1];
b=[smina;smaxa];
Ksa=inv(a)*b;
%escalamiento del acelerometro al segundo motor
lmina=-25;
lmaxa=25;
smina2=19;
smaxa2=11;
a=[lmina 1;lmaxa 1];
b=[smina2;smaxa2];
Ksa2=inv(a)*b;
%escalamiento del radiocontrol canal 3 Thrust
lmin=4292178000;%rc flysky
lmax=4292334000;%rc flysky
smin=0;
smax=2;
c=[lmin 1;lmax 1];
d=[smin;smax];
Ks=inv(c)*d;
%observador de integrador
num=1;
den=[1 0];
[A,B,C,D]=tf2ss(num,den);
po=[-0.4];
Ht=place(A',C',po);
H=Ht';
Ao=A-H*C;
Bo=[B H];
Co=1;
Do=zeros(1,2);
MFTo=zpk(ss(Ao,Bo,Co,Do));
[ao,bo,co,do]=c2dm(Ao,Bo,Co,Do,Ts,'zoh');
%controlador
nc=[1.0000 3.0000 2.2500];
dc=[1.0000 60.0700 4.2000];
c=tf(nc,dc);
cd=c2d(c,Ts);
gain=9.5;
[ncl,dc1]=tfdata(cd,'v')

```


2.5.9 Monitoreo de variables

Las variables se monitorean a través del Code Composer Studio (CCS) por medio del puerto USB poder observar el comportamiento que tienen.

Para monitorear las mediciones de los sensores, el error, la entrada al sistema o las salidas de los motores en el CCS se necesitan definir dichas variables en el código de Simulink. Para explicar el proceso se tomará como ejemplo la medición de la posición angular estimada. Primero en la conexión de la salida del observador se escribe el nombre de la variable que en este caso es “posición”. Para definir la variable se selecciona la conexión y se escribe el nombre. Cuando ya tenga el nombre el siguiente paso es abrir el *Model Explorer*, esta opción sirve para observar todos los bloques del código con sus respectivos nombres, aquí aparece la variable definida con el símbolo tal como se aprecia en la Figura 2.37.

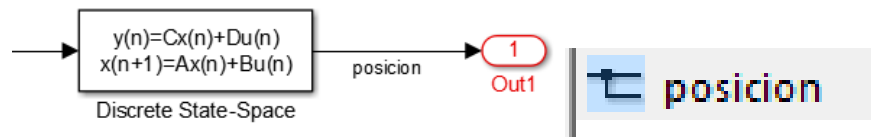


Figura 2.37 Variable a monitorear en el CCS

Al seleccionar la variable, de lado derecho aparece un menú con opciones, 3 pestañas, lo que hay que hacer es seleccionar la pestaña *Code Generation* y en *Storage class* seleccionar la opción *ExportedGlobal* y aplicar los cambios.

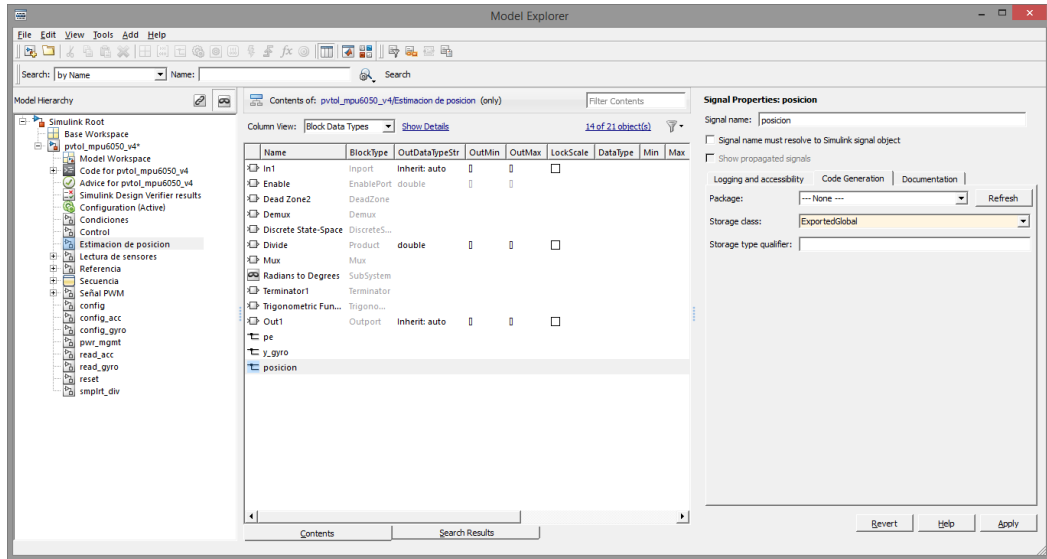


Figura 2.38 Configuración de la variable a monitorear

Este pequeño proceso se repite para cada variable a monitorear; las más importantes o que son interesantes de ver son: error, referencia, posición angular estimada, entrada U y PWM de motores.

Después de terminar de agregar los variables a monitorear y que el código esté terminado se compila el programa.

En CCS hay que hacer la conexión con el Piccolo y cargar el código para poder agregar las variables definidas. El procedimiento es seleccionar la ventana *Expressions* y activar la opción *Continuous Refresh*, el icono tiene dos flechas amarillas en sentido contrario al de las manecillas del reloj. Después hay que agregar una nueva expresión con el nombre de la variable definida que en este caso es "posición" al hacer esto automáticamente aparece información de la variable tal como se aprecia en la Figura 2.39.

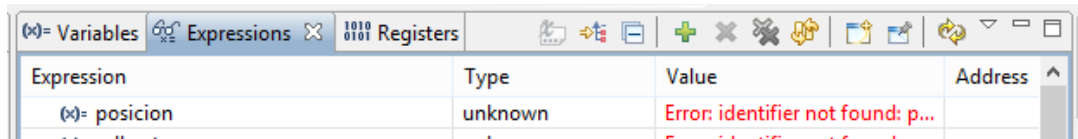


Figura 2.39 Pestaña Expressions con la variable agregada

Al ejecutar el programa las variables se actualizan según el tiempo de muestreo del CCS. El tiempo de muestreo mínimo que soporta el CCS es de 100ms pero es necesario ajustarlo ya que tiene otro valor por defecto. Para ajustar el tiempo de muestreo se abre la ventana *Preferences* que se encuentra en *Window*. Después dentro de *Code Composer Studio* seleccionar *Debug* y escribir 100 en el cuadro de texto que se llama *Continuous refresh interval* y guardar los cambios como en la Figura 2.40.

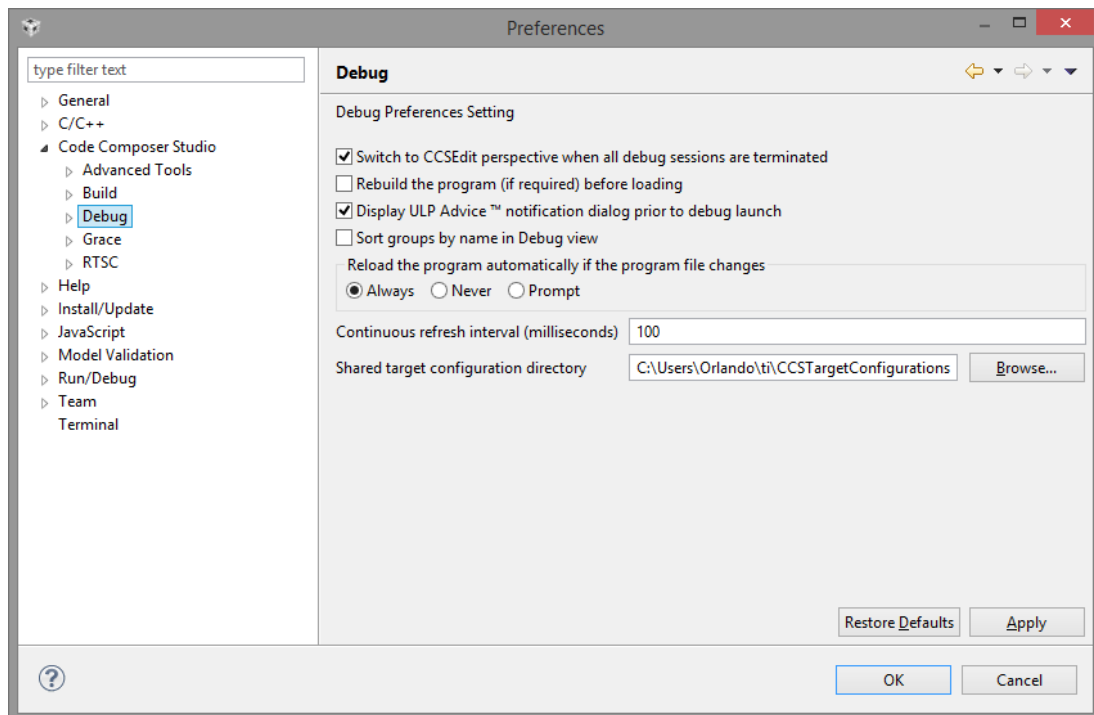


Figura 2.40 Configuración del tiempo de muestreo del CCS

Para apreciar mejor los cambios y poder almacenar la información se hace uso de las gráficas que son herramientas que incluye el CCS. Para graficar una variable existen dos maneras. La opción simple es seleccionar la variable a graficar y con la opción *Graph* y se genera la figura con las características por defecto. La segunda forma es generar la figura con las características deseadas. Esta segunda opción es seleccionando en *Tools* el tipo de gráfica, es decir, con una sola variable o para dos variables, en el caso de las dos variables es muy útil porque tienen la misma escala de tiempo. Las características se modifican en la

ventana *Graph Properties*, ventana que aparece luego de seleccionar el tipo de gráfica, son: *Acquisition Buffer Size* = 1, *Dsp Data Type* = según el tipo de dato de la variable, *Sampling Rate Hz* = 10 Hz para que la escala de tiempo coincida con el tiempo real, *Start Address* = variable a graficar pero con el símbolo “&” al inicio del nombre, por ejemplo, &posicion, *Display Data Size* = depende de las muestras que se deseen mostrar en la gráfica y que se quieran almacenar, *Time Display Unit* = segundos. La Figura 2.41 muestra las configuraciones para graficar la variable posición.

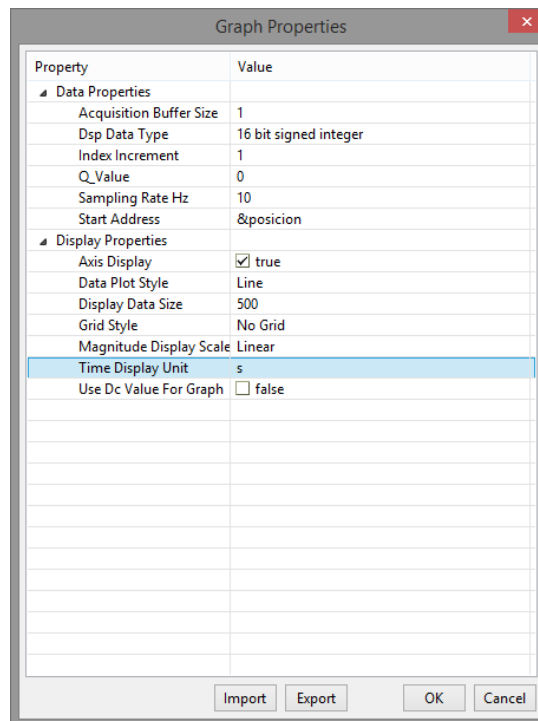


Figura 2.41 Configuración para graficar una variable

Para almacenar los datos se tiene que desactivar la opción *Continuos Refresh* y sobre la gráfica se da clic derecho y se selecciona en *Data* la opción *Export All...*, la desventaja que tiene es que sólo se pueden almacenar los datos mostrados en el la figura por eso es importante pausar la gráfica.

También es posible monitorear y modificar los valores de las constantes que están en el código, se agregan al CCS de igual manera que las variables pero

tienen una estructura en la sintaxis. La ventaja de agregar una constante en este caso es poder cambiar el tipo de referencia del sistema e incluso para modificar la ganancia del controlador sin tener que desconectar y reprogramar el Piccolo.

Tomando como ejemplo el selector de la referencia se renombra el bloque de la constante como *sel_ref*, por lo tanto cada constante que se desee monitorear y/o modificar desde CCS se tiene que renombrar.

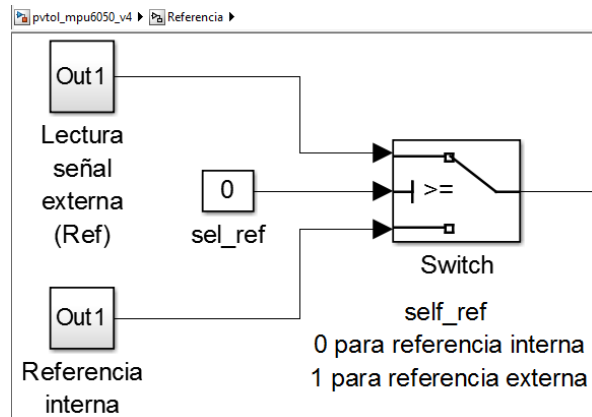


Figura 2.42 Código para seleccionar la fuente de la referencia

El nombre de la expresión se difiere al de una variable porque al agregar la variable se utiliza el mismo nombre que se define en Simulink sin embargo en el caso de las constantes tiene la siguiente estructura:

noma_P.nomc_Value

Donde *noma* es el nombre del archivo de Simulink y *nomc* el nombre de la constante, por ejemplo si el nombre del archivo de Simulink es *pvtol_mpu6050_v4* y la constante es *sel_ref* la expresión que se agrega es:

pvtol_mpu6050_v4_P.sel_ref_Value

El valor por defecto que tiene la constante se define en Simulink. Durante la ejecución del programa en CCS se pueden modificar los valores de las constantes pero si se resetea o se reprograma el Piccolo los cambios no se guardan ya que toma el valor por defecto definido.

2.6 Resultados Experimentales

El Controlador (14) y el observador (13) fueron discretizados utilizando la transformada Z con un retenedor de orden cero e implementados en el Piccolo del banco de pruebas con un tiempo de muestreo de 1kHz.

Se realizaron una serie de experimentos dentro del túnel de viento a diferentes velocidades: 0 m/s, 4 m/s y 6.5 m/s. Además, se utilizaron dos diferentes señales de referencia: una serie de escalones y una señal tipo Chirp de 0.05 Hz a 1Hz. La Figura 2.43 y la Figura 2.44 que se muestran a continuación muestran los resultados de dichos experimentos.

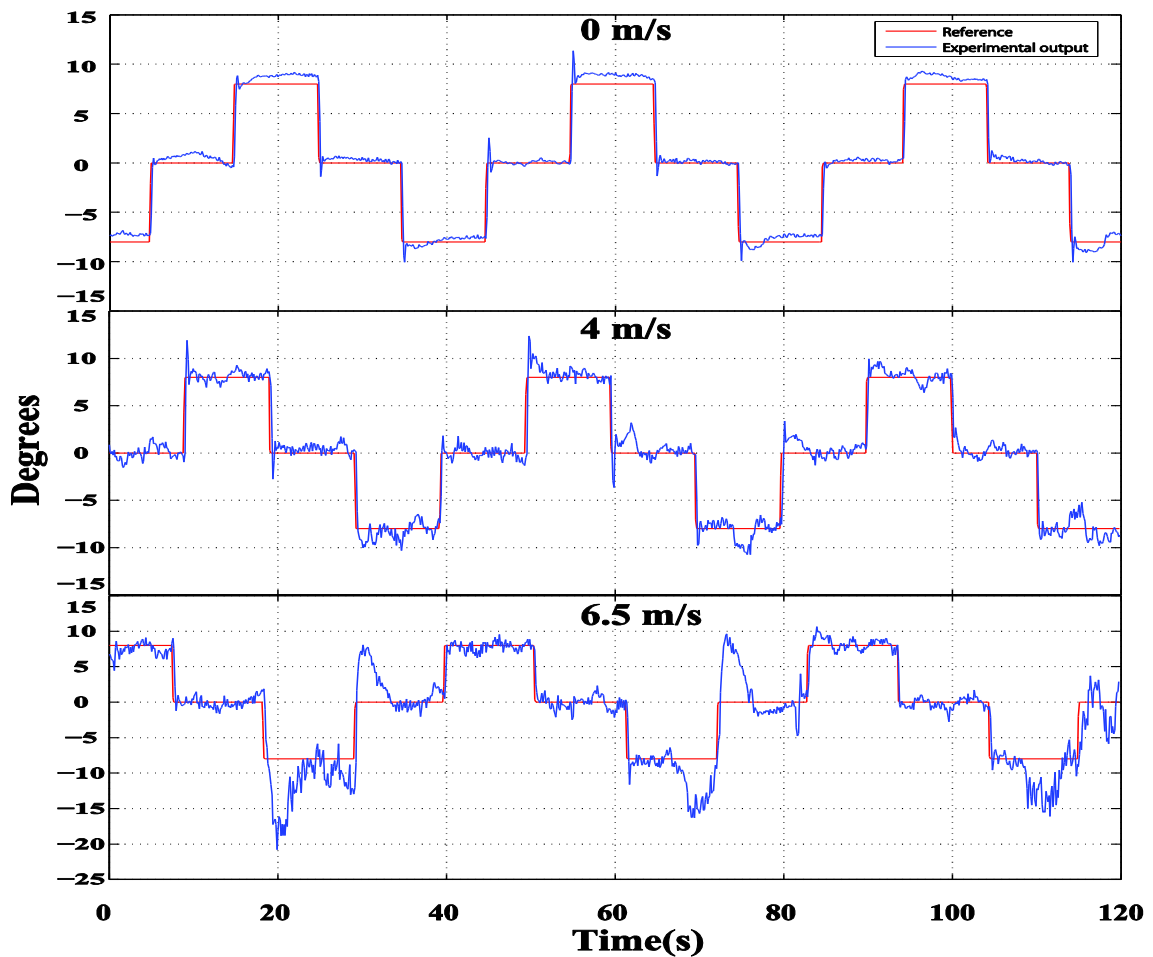


Figura 2.43 Respuesta experimental a diferentes velocidades en el túnel de viento usando escalones de referencia

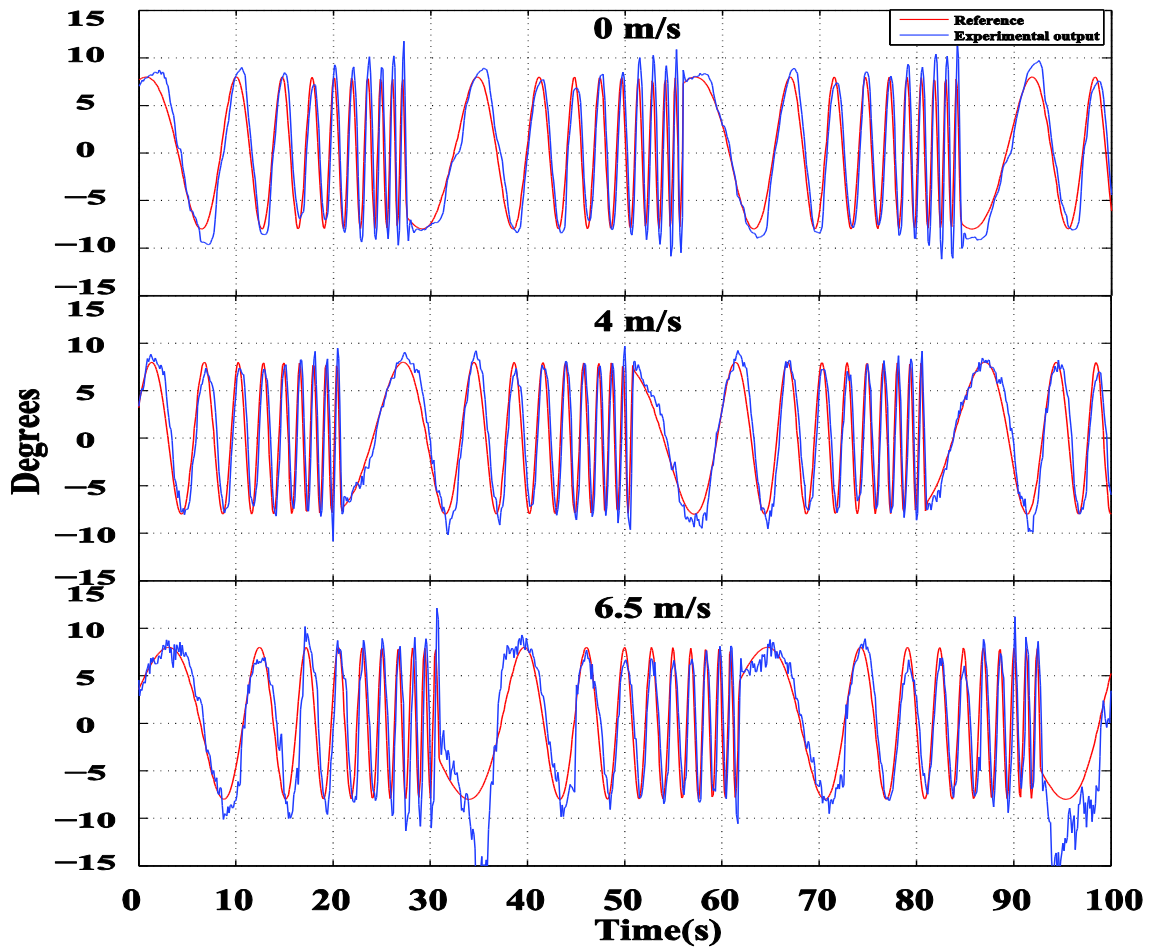


Figura 2.44 Respuesta experimental a diferentes velocidades dentro del túnel de viento usando la señal Chirp como referencia

Las respuestas experimentales permiten hacer varias observaciones cualitativas con respecto al efecto de la ráfaga de viento sobre el sistema de control.

- El nivel de ruido de las mediciones incrementa significativamente con la velocidad del viento, esto se encontró debido principalmente a las vibraciones inducidas y su efecto en los acelerómetros.
- El incremento del efecto que tiene sobre el sistema la velocidad del viento no es lineal. Esto es, de 0 m/s a 4 m/s el efecto es suave; sin embargo, un

incremento de 2.5 m/s es suficiente para deteriorar notablemente la respuesta, particularmente en el caso de las referencias de escalones.

Una observación adicional es que, considerando el análisis del IPR de la sección 2.4, hay complejidades en la forma que la ráfaga de viento interactúa con el dispositivo. En particular, en los experimentos con la serie de escalones como referencia el desempeño del sistema disminuye de una manera significativa a medida que aumenta la velocidad del viento. Recordando el IPR de la Figura 2.9, esto indica que los incrementos de la velocidad del viento están acompañados por perturbaciones en la entrada (esto es, momentos inducidos sobre el cuerpo del PVTOL) con contenido de alta frecuencia dentro de 0.1-10 rad/s. Por otro lado, en los experimentos con la señal Chirp como referencia es más fácil observar que la ráfaga de viento puede ser rechazada mejor cuando el PVTOL opera alrededor de las frecuencias medias, incluso cuando la velocidad más alta está presente. Esto sugiere que la interacción de la frecuencia del ángulo de alabeo y la ráfaga de viento de hecho reduce los momentos inducidos alrededor de la frecuencia del ángulo de alabeo. Además, el efecto de la ráfaga de viento no depende sólo de la posición instantánea angular de alabeo, sino también de todo el contenido de la frecuencia. Esta discusión muestra que la caracterización de la ráfaga de viento como una perturbación de entrada no es directa y requiere de un modelo más complejo para ser capaz de predecir estos efectos. Estos efectos no pueden ser pronosticados sin análisis y experimentación adicionales.

Un ensayo más cuantitativo del efecto de la velocidad del viento sobre el comportamiento fue hecho calculando la media cuadrática (RMS) del error de seguimiento para todos los casos. Los resultados se presentan en la Tabla 2-4. Esta tabla también confirma algunas de las características observadas con respecto a los efectos de la ráfaga de viento sobre el sistema. Primero, el incremento del error RMS tiene una relación no lineal con la velocidad del viento. Además, el efecto de la ráfaga de viento no depende sólo de la velocidad del viento sino también de la frecuencia a la que se mueve el ángulo del vehículo.

Tabla 2-4 Media cuadrática (RMS) del error de seguimiento

	0 m/s	4 m/s	6.5 m/s
Escalones	1.0497	1.7083	2.873
Chirp	2.266	2.154	3.1355

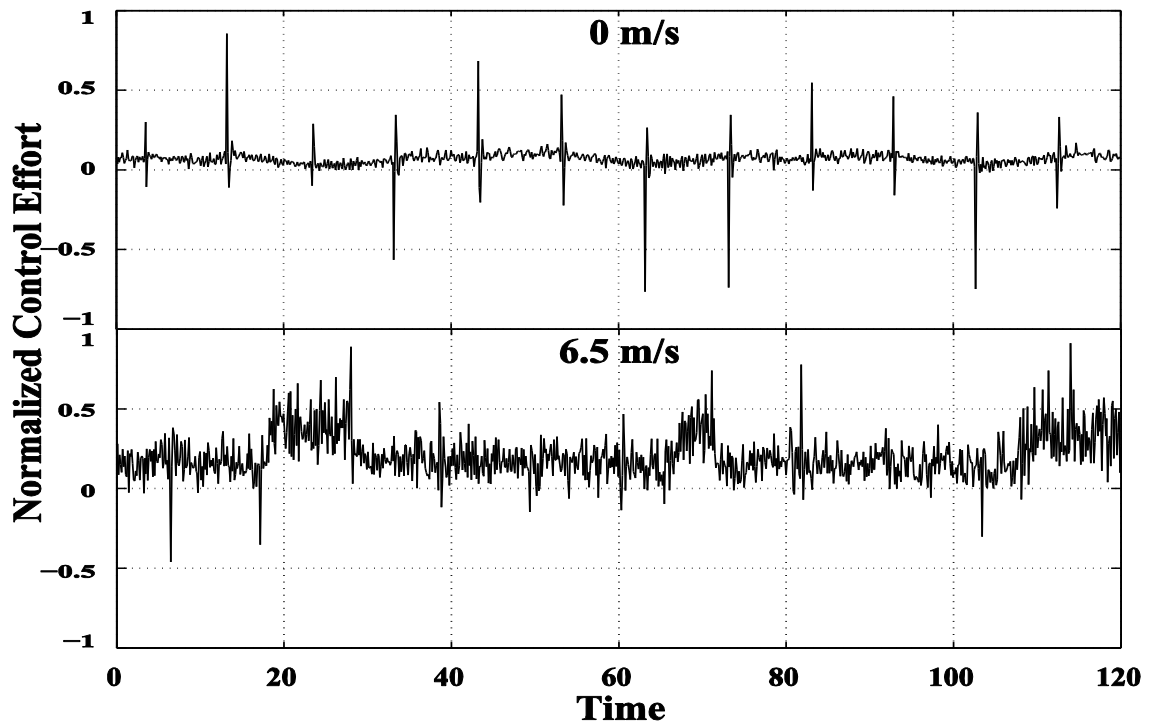


Figura 2.45 Esfuerzo de control normalizado en los experimentos de la Figura 2.43

Finalmente, la Figura 2.45 presenta el esfuerzo de control a 0 m/s y 6.5 m/s para los experimentos con la referencia de la serie de escalones. El rango de operación del esfuerzo de control es dentro del rango $[-1, 1]$; por lo tanto, el sistema no incurre en la saturación del actuador. Además, está claro que para las altas velocidades del viento el promedio del esfuerzo de control incrementa. Por otra parte, el hecho de que a 0 m/s el promedio del esfuerzo de control no sea cero sugiere que uno de los motores genera más fuerza de empuje que el otro con el mismo nivel de PWM. Finalmente, en el experimento a 0 m/s se observa que el esfuerzo de control sigue un patrón sugestivo del stick-slip; mientras el

ángulo de alabeo esta “atorado” o moviéndose ligeramente sólo (ver Figura 2.43 a 0 m/s) el esfuerzo de control sigue un patrón triangular durante el periodo constante de la referencia de escalones [12]. A mayor velocidad de viento es más difícil apreciar este fenómeno.

3 Cuadri-rotor

3.1 Modelado del sistema

Para obtener el modelado se considera al cuadri-rotor como un cuerpo rígido, además se utilizan marcos de referencia mixtos. Sin embargo al utilizar estos marcos de referencia se añaden ecuaciones innecesarias al modelo que complican el proceso de diseño del controlador. Por lo cual, es necesario obtener un modelo del sistema más simplificado que sea más adecuado para el diseño de los controladores.

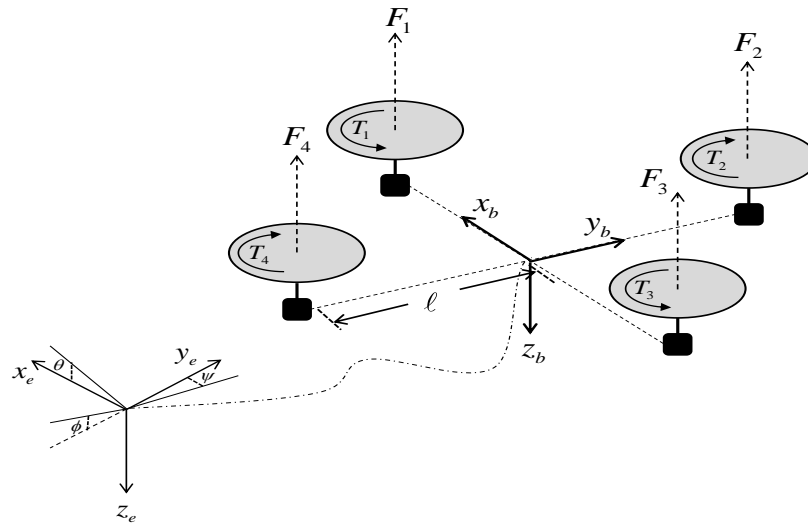


Figura 3.1 Marco de referencia del cuadri-rotor y configuración de las hélices [14]

El modelado se puede hacer a través de la dinámica de cuerpo rígido rotacional y traslacional en un marco de referencia inercial como se muestra en la Figura 3.1. Las ecuaciones de movimiento de Newton-Euler están dadas por:

$$m\dot{V}_b + m\omega_b \times V_b = F_b \quad (15)$$

$$J\dot{\omega}_b + \omega_b \times (J\omega_b) = M_b \quad (16)$$

Donde $V_b = [uvw]^T$ y $\omega_b = [p_b q_b r_b]^T$ son los vectores de velocidades lineales y angulares, F_b es el vector de fuerzas externas, m es la masa, J la matriz de momentos de inercia y M_b es el vector de momentos externos.

Para aplicaciones de seguimiento de trayectoria se opta por expresar las velocidades lineales en un marco de referencia terrestre denotado como $V_e = [\dot{x} \dot{y} \dot{z}]^T$. Teniendo en cuenta los ángulos de Euler $\Omega = [\phi \theta \psi]^T$ con secuencia $\psi - \theta - \phi$ (guiñada-cabeceo-alabeo), V_e se expresa como:

$$V_e = R^T V_b \quad (17)$$

Donde R es la matriz de rotación sobre los ángulos $\psi - \theta - \phi$:

$$R = \begin{bmatrix} c_\theta c_\psi & c_\theta s_\psi & -s_\theta \\ c_\psi s_\theta s_\phi - s_\psi c_\phi & s_\psi s_\theta s_\phi + c_\psi c_\phi & c_\theta s_\phi \\ c_\psi s_\theta c_\phi + s_\psi s_\phi & s_\psi s_\theta c_\phi - c_\psi s_\phi & c_\theta c_\phi \end{bmatrix} \quad (18)$$

Donde $s_x = \sin(x)$ y $c_x = \cos(x)$. De manera similar, las derivadas de los ángulos de Euler se pueden obtener a partir de las velocidades angulares del marco de referencia del cuerpo ω_b de acuerdo con la siguiente relación:

$$\omega_b = \dot{\Phi} + R_\phi \dot{\Theta} + R_\phi R_\theta \dot{\Psi} = R_\alpha \dot{\Omega} \quad (19)$$

Donde:

$$R_\alpha^{-1} = \begin{bmatrix} 1 & s_\phi t_\theta & c_\phi t_\theta \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi / c_\theta & c_\phi / c_\theta \end{bmatrix}, \quad \Phi = [\phi \ 0 \ 0]^T \\ \Theta = [0 \ \theta \ 0]^T \\ \Psi = [0 \ 0 \ \psi]^T$$

y $t_x = \tan(x)$. Las ecuaciones (15), (16), (17) y (19) representan el modelo matemático del sistema de 12 estados sin tener en cuenta las fuerzas aerodinámicas.

La fuerza de sustentación de la hélice se puede aproximar como $F_i = k_p V_i^2$, donde V es el voltaje que se le aplica al motor y k_p una constante que se puede caracterizar experimentalmente para cada combinación de motor y hélice. De

igual manera se puede aproximar el momento reactivo de una sola hélice como $T_i = k_m V_i^2$, donde k_m es una constante que se puede caracterizar de la misma manera que a k_p .

Existen otras fuerzas y momentos inducidos por la hélice que son: el blade flapping, las fuerzas giroscópicas en cada hélice y el lead-lag. Sin embargo, se ha observado que dichos efectos no tienen gran importancia debido a la simetría con la que operan las hélices.

De la Figura 3.1 se establece la siguiente configuración para que las hélices puedan producir momentos de fuerzas y las fuerzas deseadas:

$$\begin{bmatrix} V_1^2 \\ V_2^2 \\ V_3^2 \\ V_4^2 \end{bmatrix} = P \begin{bmatrix} U_z \\ T_p \\ T_q \\ T_r \end{bmatrix} = \begin{bmatrix} 1/4k_p & 0 & 1/2\ell k_p & -1/4k_m \\ 1/4k_p & -1/2\ell k_p & 0 & 1/4k_m \\ 1/4k_p & 0 & -1/2\ell k_p & -1/4k_m \\ 1/4k_p & 1/2\ell k_p & 0 & 1/4k_m \end{bmatrix} \begin{bmatrix} U_z \\ T_p \\ T_q \\ T_r \end{bmatrix} \quad (20)$$

Donde la matriz P se puede considerar como una base ortogonal, por lo tanto, los efectos de las entradas $[U_z \ T_p \ T_q \ T_r]$ son independientes y definidos con los voltajes de entrada.

En [14] se muestra un procedimiento para obtener un modelo simplificado aproximado del cuadri-rotor, el cual resulta en la siguiente matriz función de transferencia:

$$G(s) = \text{diag} \left[\frac{g}{I_\alpha s^2} \quad \frac{g}{I_\alpha s^2} \quad \frac{1}{I_z s^2} \right] \quad (21)$$

$$\begin{bmatrix} \theta(s) \\ \phi(s) \\ \psi(s) \end{bmatrix} = G(s) \begin{bmatrix} T_q(s) \\ T_p(s) \\ T_r(s) \end{bmatrix} \quad (22)$$

La cual se complementa con la matriz de propulsión de la ecuación (20), de la cual se utiliza solamente el vector de entrada $[T_q \ T_p \ T_r]$. En las siguientes secciones de esta tesis se utiliza el modelo simplificado.

Las constantes del modelo (22) se determinaron siguiendo un proceso similar al que se siguió en la sección 2.2 para el PVTOL, de tal manera que el modelo de diseño resultante es igual a:

$$\begin{bmatrix} \theta(s) \\ \phi(s) \\ \psi(s) \end{bmatrix} = \text{diag} \left[\frac{6.7255}{s^2} \quad \frac{6.7255}{s^2} \quad \frac{0.6859}{s^2} \right] \begin{bmatrix} U_\theta(s) \\ U_\phi(s) \\ U_\psi(s) \end{bmatrix} \quad (23)$$

donde U_θ , U_ϕ y U_ψ son entradas de control que operan en un rango de -500 a 500 de tal manera que en condiciones de equilibrio (esto es, “hover”) -500 implica apagar el motor, 500 operar el motor a la máxima velocidad y cero implica hover. Las entradas de control son adimensionales ya que el Crius depende de éstas para generar las señales PWM.

3.2 Banco de pruebas

Para realizar los experimentos previos a los correspondientes al vuelo se diseñaron dos bases: una con movimiento en el eje de la guiñada y la otra con movimiento en los tres ejes rotacionales.

El propósito por el cual se diseñó el primer banco de pruebas fue para ajustar el controlador de la guiñada.

El segundo banco de pruebas se desarrolló para realizar los experimentos con los tres ejes liberados y ajustar los controladores. Además, sirve como antesala a los experimentos en vuelo libre y así evitar algún accidente y daño mayor a los componentes ya que sólo es cuestión de ajustar las ganancias.

3.2.1 Banco para ángulo de guiñada

Consta de tres piezas: las piezas A1 y A2 de la Figura 3.2 fueron fabricadas por medio del prototipado rápido (impresión 3D), la tercera pieza es un balero. La pieza A2 se monta sobre un perfil Bosch 20x20 y sobre la pieza A1 se coloca el cuadri-rotor.

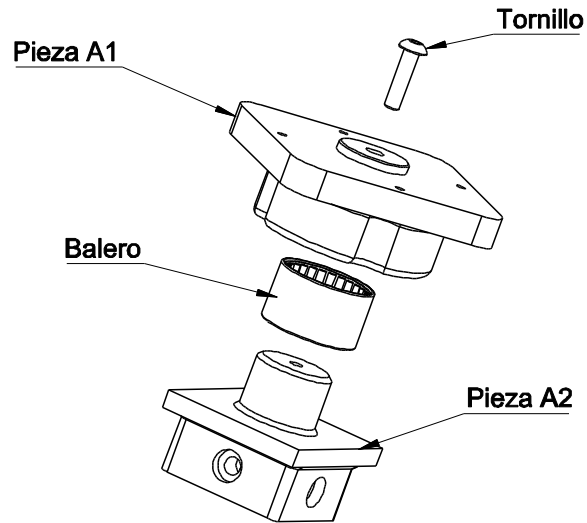


Figura 3.2 Ensamble de la base con movimiento en guiñada

Los planos de las piezas A1 y A2 se encuentran en los anexos de esta tesis.

3.2.2 Banco de pruebas con movimiento en tres ejes

Consta de cuatro piezas: tres de estas la B1, B2 y B3 fueron fabricadas por medio de la impresión 3D y la última es una rótula. La pieza B3 se coloca sobre un perfil Bosch 20x20 y sobre la pieza B1 se fija el cuadri-rotor.

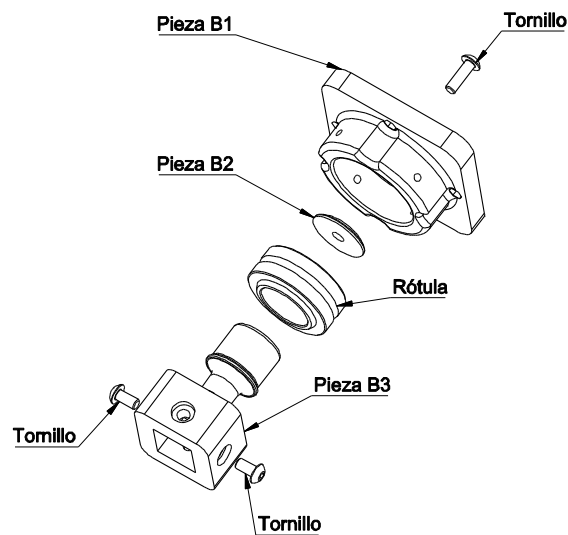


Figura 3.3 Ensamble de la base con movimiento en los 3 ejes

Al igual que en la sección anterior los planos de las piezas B1, B2 y B3 se encuentran dentro de los anexos.

3.3 Diseño del observador

Al igual que lo visto en la sección 2.3, donde se diseñó el observador para el movimiento de alabeo, se va a seguir el mismo concepto, es decir, el observador crea un filtro complementario. En este caso se van a diseñar dos observadores uno para la estimación de alabeo y cabeceo y el otro para la guiñada. Para el alabeo y cabeceo siguen utilizando las mediciones del acelerómetro y giroscopio en sus respectivos ejes, mientras que para la guiñada se utiliza al giroscopio y magnetómetro.

El criterio por el cual se seleccionó el polo del observador fue basándose en el desempeño que fue observado del cuadri-rotor durante diversos experimentos; se tomó la decisión de ajustarlo de esta manera debido a la ausencia de un modelo matemático del comportamiento de los sensores. El efecto que tiene este polo se nota en la Figura 3.4 donde se aprecia la respuesta del observador con diferentes polos. Estas respuestas se obtuvieron procesando los datos crudos de los sensores fuera de línea con el observador en cuestión.

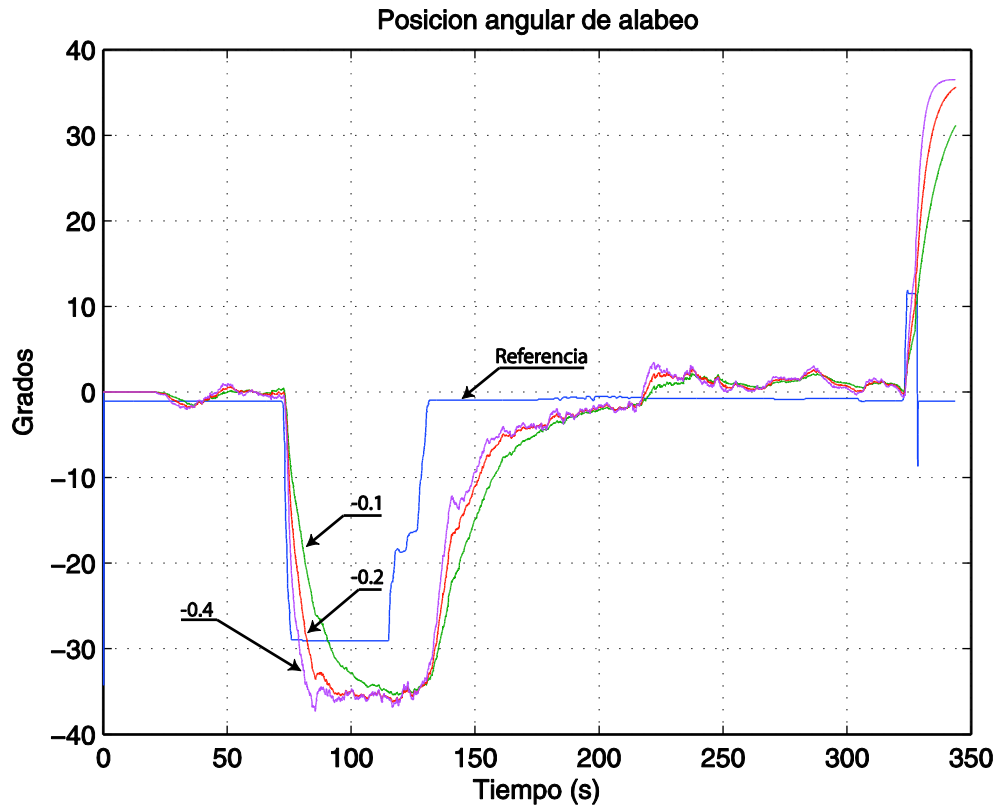


Figura 3.4 Comparación de la salida observada que tiene el sistema al colocar el polo del observador en -0.1, -0.2 y -0.4.

En este experimento, la línea azul representa la referencia de un esquema de control simple (cuyo diseño se discute más adelante), mientras que las otras líneas son las salidas estimadas con el observador. Al final se seleccionó el polo en -0.1 por la suavidad que presenta en la respuesta, de tal manera que se espera que éste sea más robusto aunque con menor desempeño.

En el caso del observador para estimar la posición de la guiñada, se siguió un procedimiento similar y se ajustó el polo del observador en -0.8. Esto indica que la velocidad de respuesta del magnetómetro es menor en comparación a la de los acelerómetros. De tal manera, que el observador de posición en guiñada considera en mayor grado al giroscopio.

3.4 Diseño de los controladores

Los controladores se diseñaron tomando en cuenta el modelo simplificado propuesto (23) de la sección 3.1 utilizando las técnicas clásicas de Bode y las siguientes especificaciones de diseño:

- Ancho de banda 2-10 rad/s según el desempeño observado.
- Margen de fase mayor a 60°.
- Margen de ganancia mayor a 12dB.
- Controladores estables y de fase mínima.

Los controladores propuestos son los siguientes:

$$C_1(s) = \frac{(s+1.5)(s+0.25)}{(s+60)(s+0.07)} 30 \quad (24)$$

$$C_2(s) = \frac{(s+1.5)(s+0.1)}{(s+60)(s+0.07)} 784.8 \quad (25)$$

donde $C_1(s)$ se utiliza en los ejes de alabeo y cabeceo mientras que $C_2(s)$ es para el eje de la guiñada. Estos controladores se basan en la estructura del que se diseñó en el capítulo anterior para el PVTOL.

Considerando el modelo (23) de la sección 3.1 y los dos controladores propuestos las respuestas en frecuencia del sistema nominal (sólo incluye la planta y el

controlador correspondiente según el eje) en lazo abierto se muestra en las siguientes gráficas:

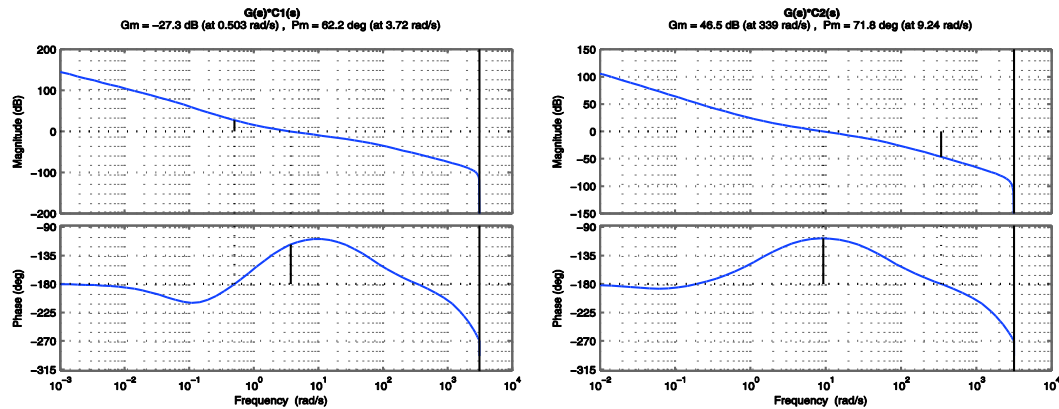


Figura 3.5 Respuesta en frecuencia del sistema en lazo abierto para los controladores C1 y C2

Es esta figura se observa que los controladores (24) y (25) cumplen con las especificaciones de diseño. En particular se observó que el eje de guiñada respondía mejor con un mayor ancho de banda (9.24 rad/s), mientras que los otros ejes respondieron mejor con menor ancho de banda (3.72 rad/s).

La respuesta en el tiempo así como la respuesta en frecuencia del sistema nominal en lazo cerrado se muestran en las siguientes figuras.

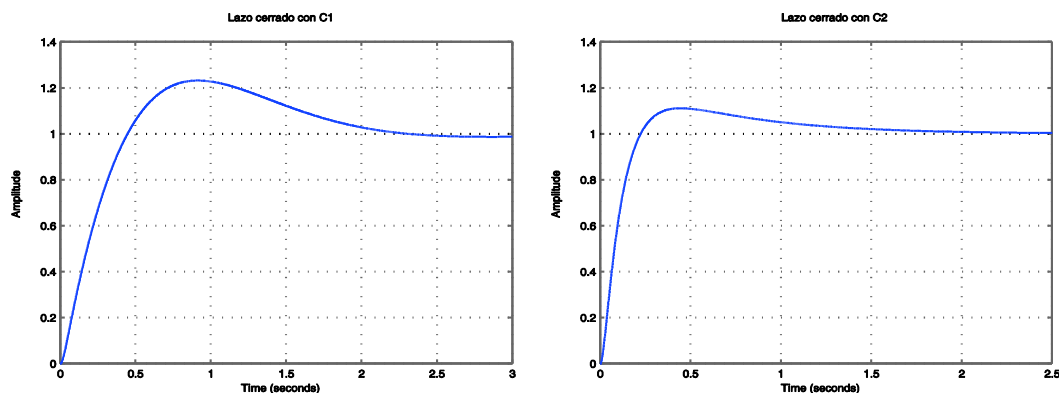


Figura 3.6 Respuesta en el tiempo del sistema en lazo cerrado ante una entrada escalón para cada controlador

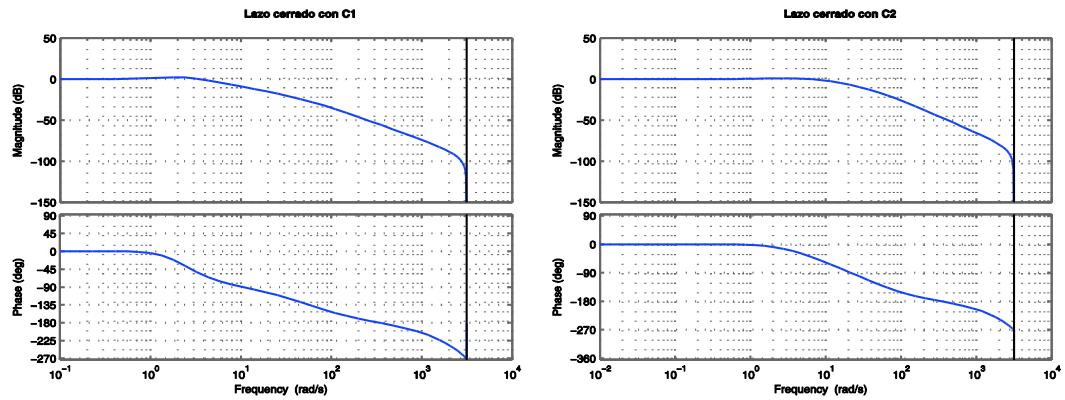


Figura 3.7 Respuesta en frecuencia del sistema en lazo cerrado para cada controlador

Se realizó un análisis de sensibilidad del sistema nominal en el cual se obtuvieron los siguientes tipos de sensibilidad: El error que tiene la respuesta del sistema con respecto a la referencia llamada sensibilidad (**S**), la sensibilidad complementaria de control (**S_c**), es decir, como responde la entrada con respecto al ruido del sensor y la sensibilidad con respecto a la entrada (**Y_u**), es decir,

determinar cuánto error hay si se tiene una perturbación a la entrada (fuerza externa) se muestran en las gráficas de la siguiente figura.

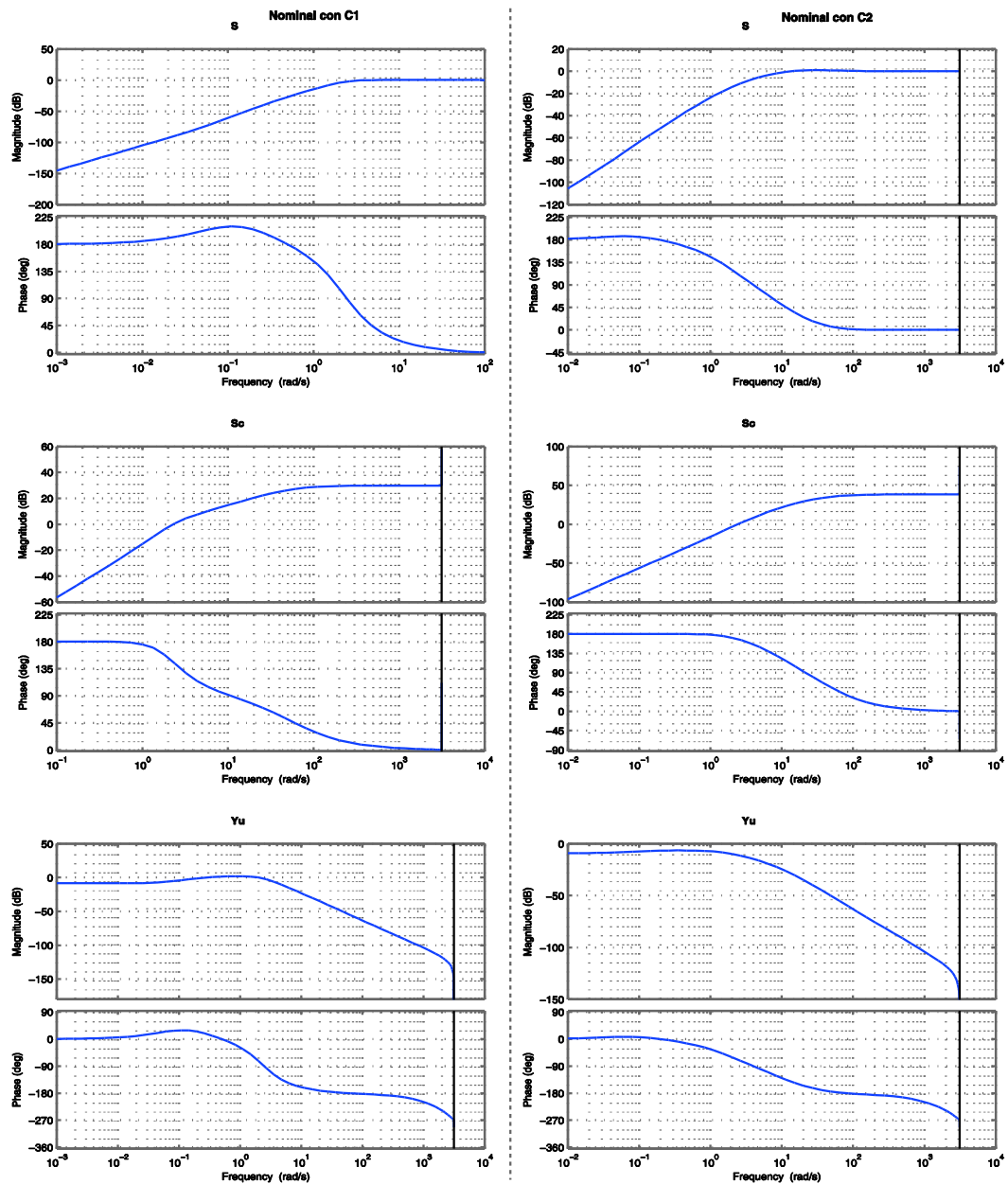


Figura 3.8 Respuestas en frecuencia de las diferentes sensibilidades que tiene el sistema en lazo cerrado nominal para ambos controladores

En estas figuras, se observa que las respuestas de sensibilidad y rechazo a perturbaciones son buenas para la sensibilidad S. No obstante, en el caso de Yu

se puede ver que el sistema no es capaz de rechazar muy bien perturbaciones a la entrada. Esto se debe a la alta ganancia del sistema mismo en lazo abierto para bajas frecuencias (un doble integrador). Esto indica que se tendrá un alto grado de sensibilidad a desbalances y otras perturbaciones de fuerzas externas. Es posible reducir esta sensibilidad incrementando la ganancia del controlador a través de un efecto integral, no obstante, esto implica mayor pérdida de fase y menor grado de robustez.

Por otro lado, el diagrama de Bode de S_c muestra que el controlador no es capaz de filtrar ruidos de alta frecuencia. Aunque el lazo de control completo no se ve en teoría afectado por esto debido a la baja ganancia de la planta misma en altas frecuencias (un doble integrador), los actuadores (en este caso los motores y sus electrónicas de potencia) no son capaces de operar con entradas de alta frecuencia. Debido a esto, durante la implementación de los controladores se observaron vibraciones generadas por los motores y fallo en su operación. Además, dichas vibraciones afectan a las mediciones de los sensores, principalmente al acelerómetro. Para mejorar el nivel de rechazo a ruido en los actuadores se incluye un filtro; el cual también afectará la respuesta en el tiempo y a la frecuencia del sistema. El filtro propuesto es el siguiente:

$$F_1(s) = \frac{90000}{(s+300)^2} \quad (26)$$

Para determinar el efecto de este filtro se vuelven a calcular las respuestas en frecuencia del sistema completo (esto es, agregando el filtro al sistema nominal).

Los diagramas de Bode en lazo abierto para cada controlador utilizando el sistema completo se muestran en la siguiente imagen.

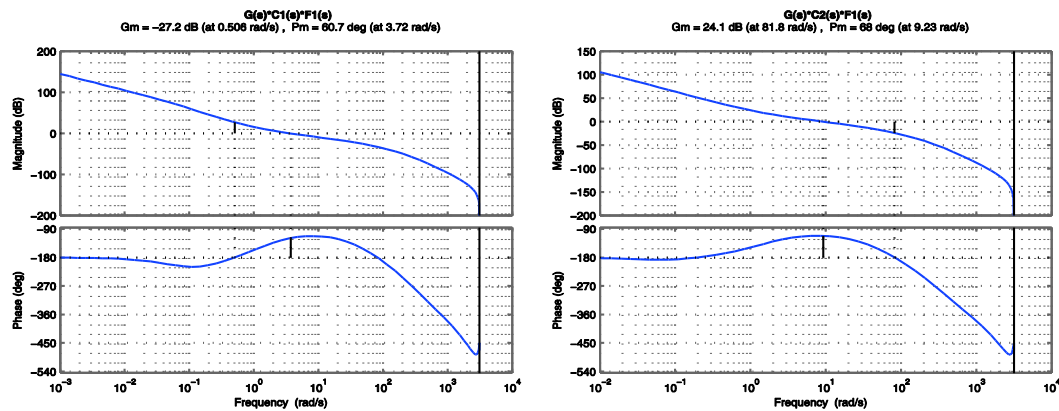


Figura 3.9 Respuesta en frecuencia de lazo abierto del sistema para cada controlador con el filtro F1

En estas figuras se ve como el margen de fase se reduce, no obstante queda dentro de las especificaciones.

La respuesta en el tiempo así como la respuesta en frecuencia del sistema completo en lazo cerrado para cada controlador se muestran en las siguientes figuras.

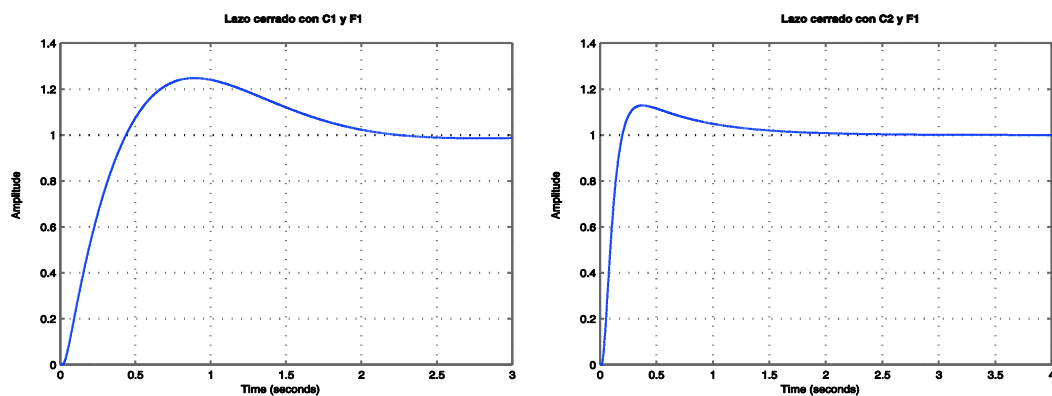


Figura 3.10 Respuesta en el tiempo del sistema en lazo cerrado ante una entrada escalón unitario para cada controlador con el filtro F1

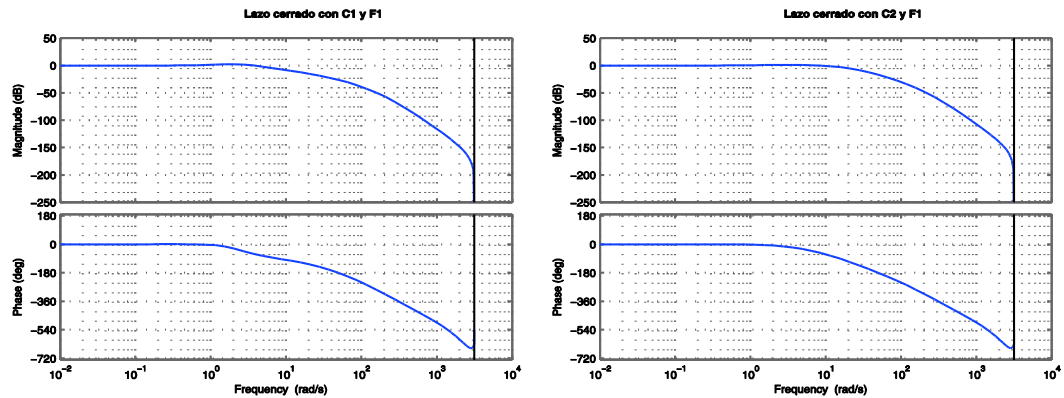


Figura 3.11 Respuesta en frecuencia del sistema en lazo cerrado para cada controlador con el filtro F1

Las gráficas de la Figura 3.12 corresponden al análisis de sensibilidad del sistema completo. Comparando estas graficas con las vistas en la Figura 3.8 se puede observar que:

De la sensibilidad S nominal el sistema tiene poco error si la referencia cambia a bajas frecuencias, generalmente el cambio de las referencias son de baja frecuencia, y la S completa prácticamente no sufre algún cambio.

En S_c es donde se puede ver claramente el efecto del filtro. En el nominal se puede ver como rechaza el ruido de baja frecuencia pero en altas frecuencias permite el paso del ruido. Al implementar el filtro, en el sistema completo, ya puede rechazar ruido de alta frecuencia.

Para Y_u al igual que paso en S la respuesta no cambia mucho al implementar el filtro. La respuesta que presenta rechaza bien las perturbaciones de alta

frecuencia pero a bajas frecuencias el rechazo a las perturbaciones de entrada continua siendo bajo.

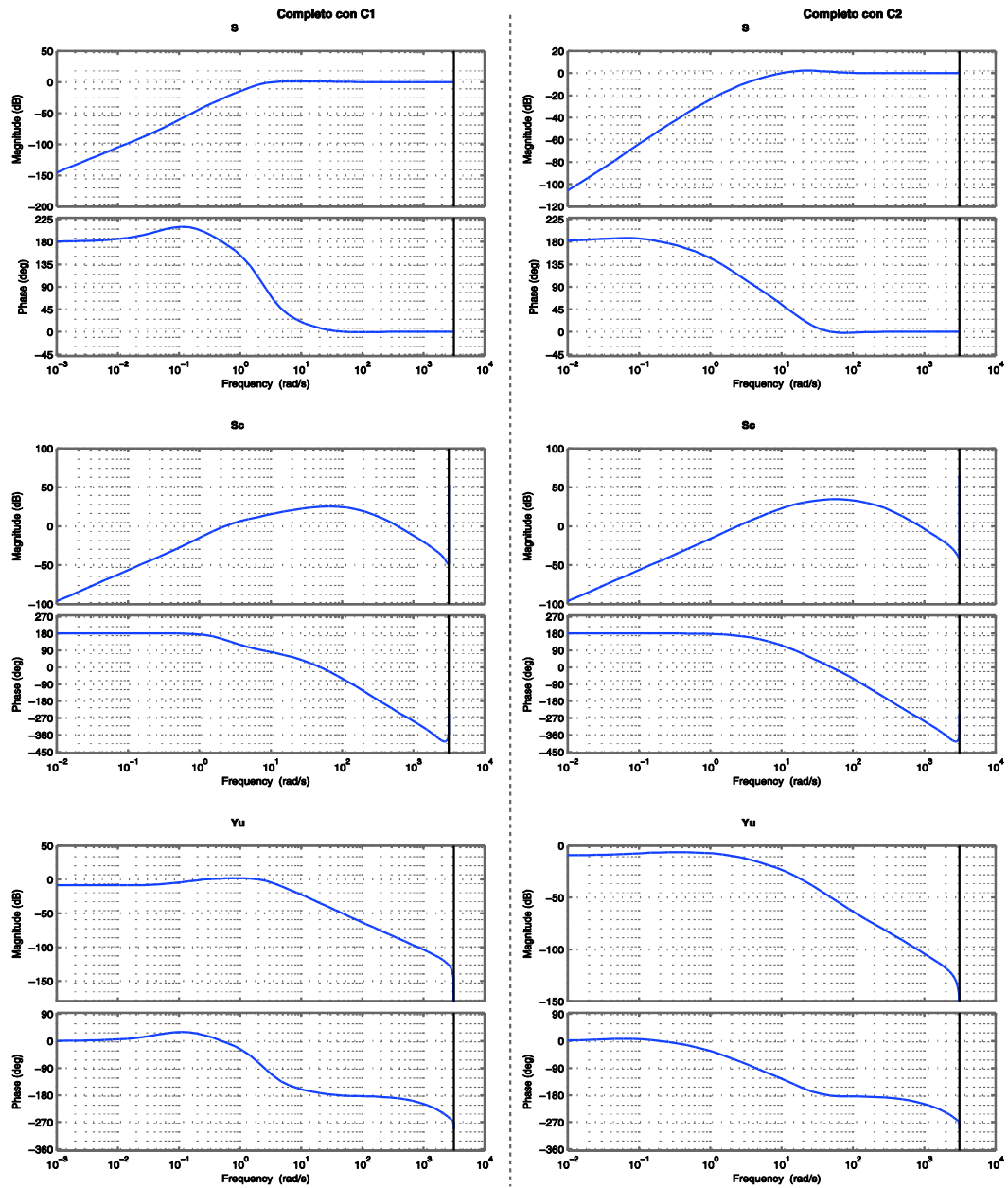


Figura 3.12 Respuesta en frecuencia del sistema en lazo cerrado completo para ambos controladores en los tres tipos de sensibilidad

3.5 Procedimiento de operación

Primero se enciende el radiocontrol, después se conecta la batería y se escucha la siguiente secuencia de tonos 123 111 1-largo donde los primeros tres tonos indican que están inicializando los controles electrónicos de velocidad, el segundo grupo de tonos indica las celdas de la batería, en este caso como la batería es de tres celdas se escuchan tres tonos y al final es un tono largo que indica que ya están listos para recibir la señal PWM.

Simultáneamente el Crius y el Piccolo se están inicializando y ambos cuentan con un led para indicar su estado. El led de color rojo del Piccolo se mantiene encendido hasta que termina de calcular y eliminar el offset de las señales de los sensores, además para asegurar que los controladores inician en cero tiene un retardo de 13 segundos. Este retardo es muy importante porque el Crius necesita una señal del radiocontrol para activar las salidas a los motores la cual se genera moviendo el canal 4 a la esquina inferior derecha hasta que el led azul del Crius se encienda. Esta operación se tiene que hacer inmediatamente cuando se apague el led rojo y es una acción rápida.

Cuando el led azul del Crius este encendido se puede conectar a la interfaz gráfica por Bluetooth a la computadora y esperar unos segundos sólo para que los primeros datos recibidos sean cero. Una vez que se conecta, la interfaz comienza a almacenar la información cruda de los sensores, así como las señales del radiocontrol, las señales que van hacia los motores, entre otras.

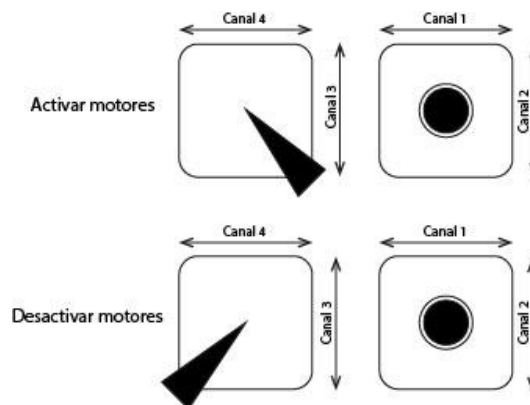


Figura 3.13 Comandos del radiocontrol para activar y desactivar los motores

Al terminar primero se desactivan las salidas PWM del Crius y se cierra la comunicación con la interfaz gráfica de la computadora para que el archivo donde se almacena la información recopilada se guarde correctamente, después se desconecta la batería y se apaga el radiocontrol.

En caso de estar configurando las ganancias de los controladores se recomienda que la comunicación por Bluetooth se haga a través de la aplicación de celular porque es más rápido modificar estos valores. Se modifican sólo las ganancias P de Roll, Pitch y Yaw según el controlador a manipular.

3.6 Descripción general del sistema

El cuadri-rotor tiene la configuración tipo X, cuenta con la computadora de vuelo (CDV) Crius v2.5, el Piccolo F28069 controlSTICK, un módulo de comunicación por Bluetooth, cuatro motores sin escobillas con sus respectivos controles electrónicos de velocidad (ESC, por sus siglas en inglés, Electronic Speed Control), 4 hélices de 10x6 y esta alimentado con una batería de 3 celdas a 2200mAh (miliamper-hora).

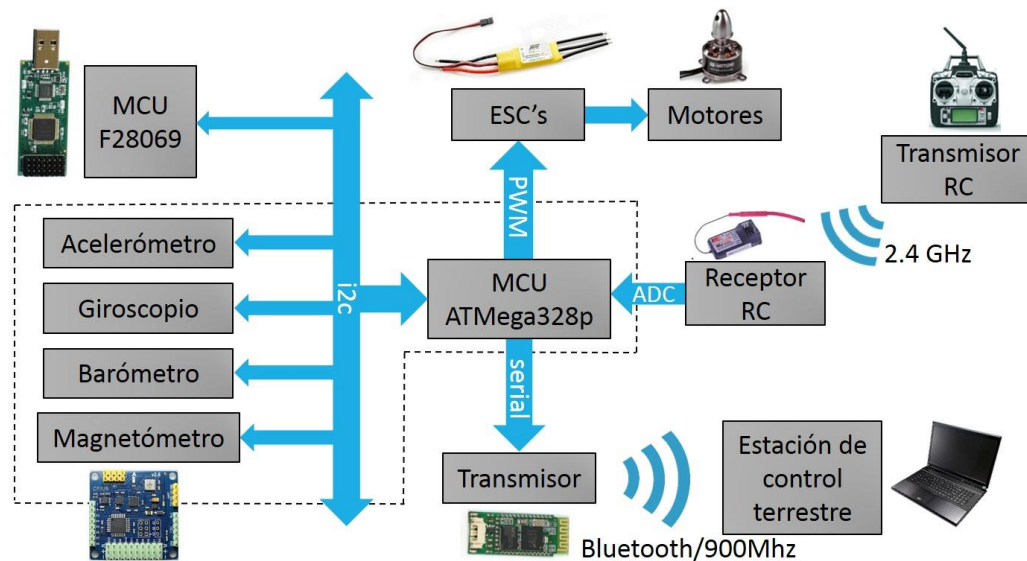


Figura 3.14 Esquema general del sistema aéreo no tripulado

El código del Crius se tuvo que modificar para que no realice el cálculo de los algoritmos de control y en su lugar transmita las señales del radiocontrol, las mediciones de los sensores y las ganancias de los controladores al Piccolo, así como de recibir por parte del Piccolo los resultados de los controladores. La comunicación entre las dos unidades de procesamiento es por medio del protocolo de comunicación I2C, donde el Crius es maestro y el Piccolo esclavo. Con la información que recibe el Piccolo se calcula una estimación de las posiciones angulares de los tres ejes, realiza el cálculo de los controladores y sincroniza las salidas de los motores. Al terminar transmite el valor que necesita el Crius para generar las señales PWM. El módulo de Bluetooth es para establecer comunicación con la computadora y almacenar las lecturas de los sensores, las entradas del radiocontrol y las salidas hacia los motores. Esta información almacenada es la que se muestra en la interfaz del Crius. Además en esta interfaz se modifican las ganancias de los controladores.

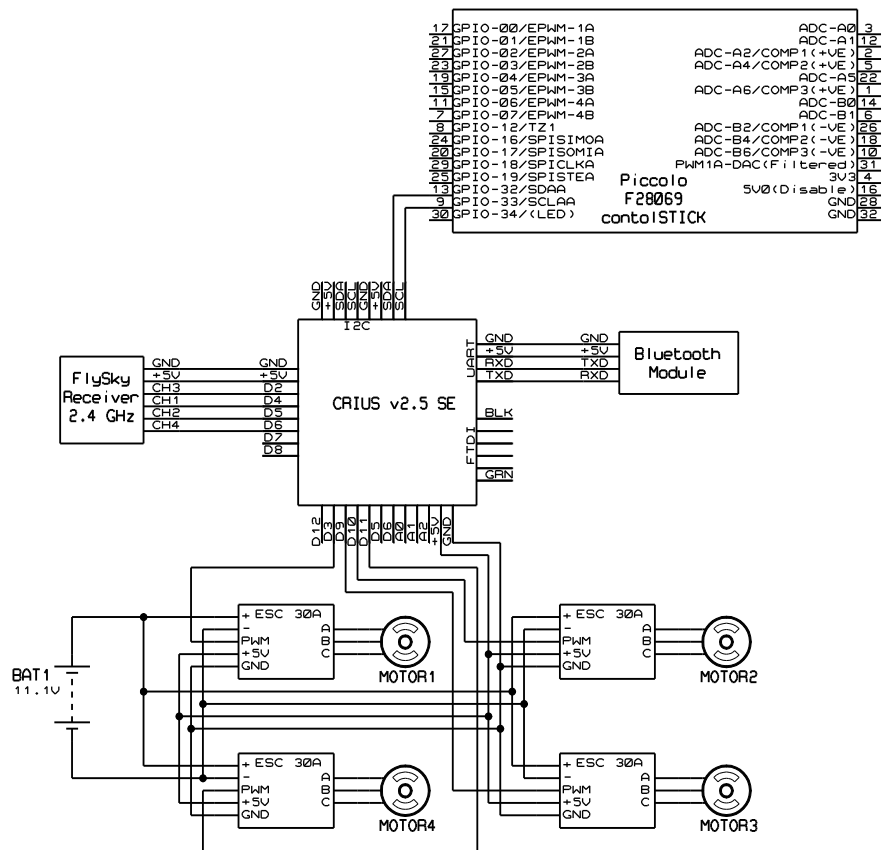


Figura 3.15 Diagrama esquemático



Figura 3.16 Cuadri-rotor

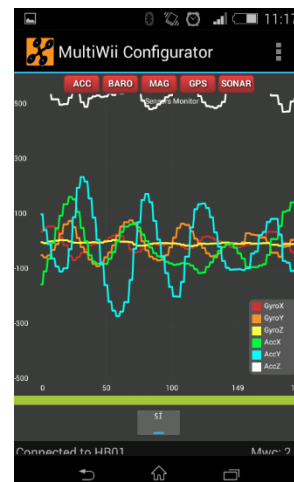
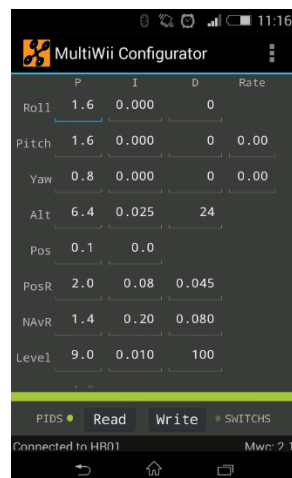
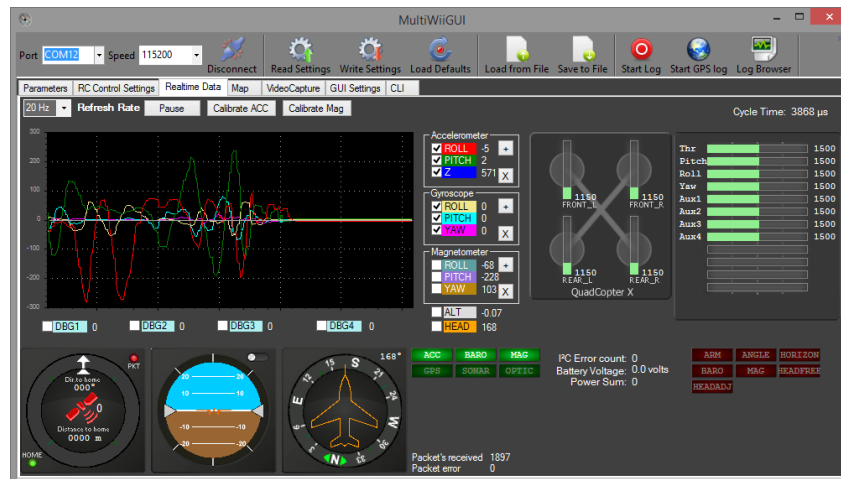


Figura 3.17 Interfaz gráfica de la estación terrestre para computadora y para celular

3.7 Programación de la computadora de vuelo.

El código generado en Simulink toma como base el código del PVTOL. El cambio más significativo es que el Piccolo funge como esclavo en la comunicación I2C. El código se divide en tres etapas, la primera es la adquisición de datos y el procesamiento de los mismos, la segunda consiste en las operaciones de los algoritmos de control y la tercera etapa consiste en transmitir por I2C los resultados obtenidos.

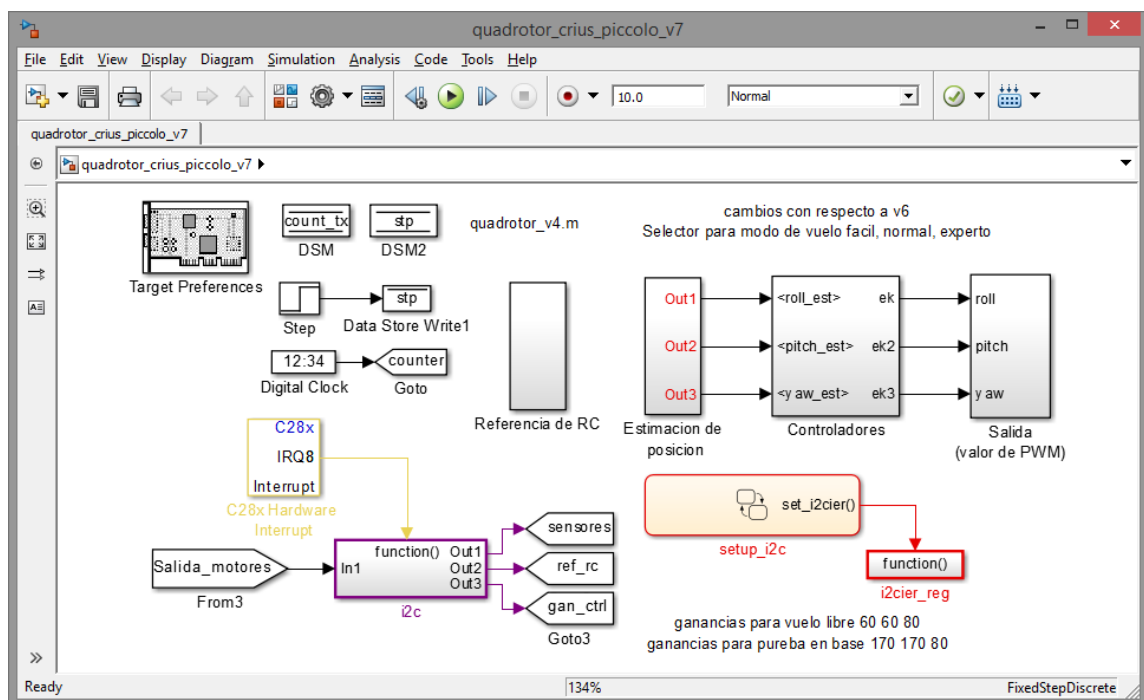


Figura 3.18 Código general del Piccolo para el cuadri-rotor

3.7.1 Protocolo de comunicación I2C

Como el Piccolo va a trabajar como esclavo en I2C se tiene que configurar para que trabaje en este modo de operación. Esta configuración se realiza en las opciones del *Target Preferences*, además se tiene que asignar la dirección que va a tener dentro del bus, la cual tiene que ser diferente a la que tienen los sensores. La dirección tiene que ser de 7 bits, en números decimales es de 0 a 127, en este caso se le asigna la dirección 10d. Por último se activan las

interrupciones del sistema ya que van a ser utilizadas para recibir y escribir datos por la comunicación I2C

3.7.2 Adquisición y procesamiento de datos

Como ahora el Piccolo está en modo esclavo no se puede utilizar la interrupción *ARDY* que fue utilizada en el código del PVTOL. Las interrupciones necesarias son *XRDY* y *RRDY* para transmitir y recibir información, sin embargo estas interrupciones no se encuentran disponibles para activar desde el *Target Preferences*. La solución o forma de poder activar dichas interrupciones es de forma manual utilizando Stateflow.

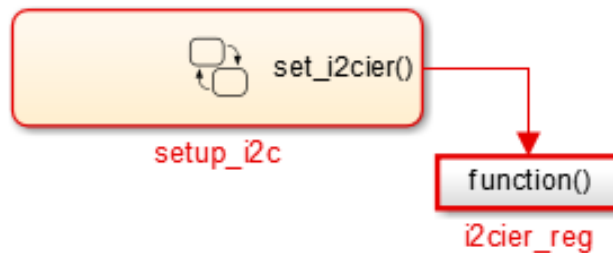


Figura 3.19 Diagrama del Stateflow para activar las interrupciones

Dentro del Stateflow se encuentran dos estados, en el primero se activa la función que activa al subsistema y el segundo estado sólo se agrega como standby.

Dentro del subsistema se encuentra el código para definir las interrupciones en el registro *I2CIER*. Este registro es usado para activar o desactivar las interrupciones, cada interrupción está almacenada en uno de los siete bits menos significativos ya que los demás bits están reservados. Para activar alguna interrupción sólo se escribe 1 en el bit correspondiente y se desactiva escribiendo 0. En este caso las interrupciones que se necesitan se ubican en los bits 3 y 4, ver tabla 14-9 de [13].

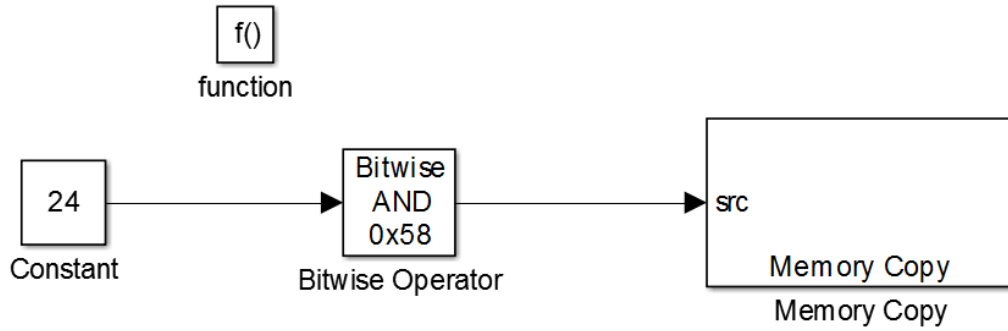


Figura 3.20 Código para activar las interrupciones

Al igual que en el código del PVTOL se agrega el apartado de las interrupciones, donde dentro del subsistema se encuentran los algoritmos correspondientes a la comunicación por I2C. Para separar la función del I2C se toma lectura del registro *I2CISRC* con el bloque *Memory Copy*, la configuración de este bloque se muestra en la Figura 3.21.

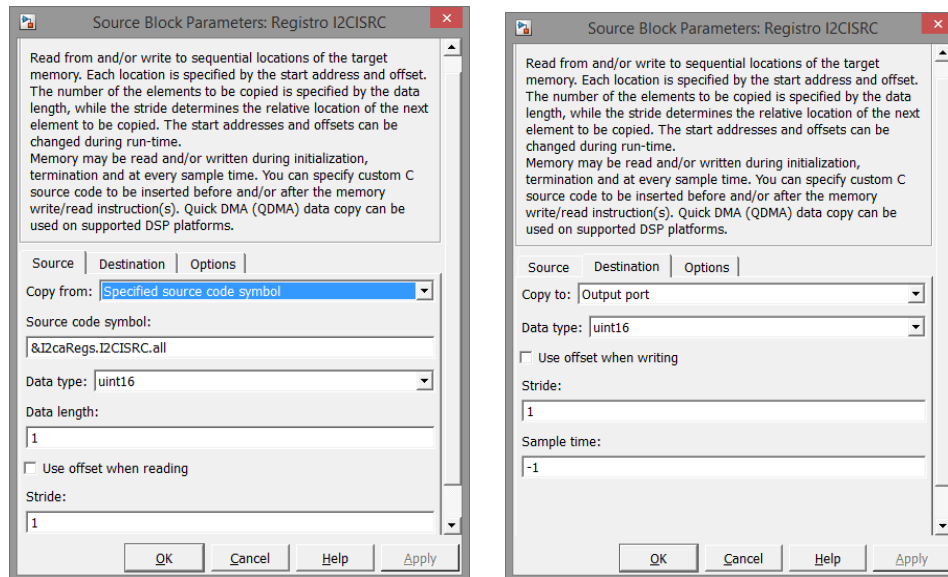


Figura 3.21 Configuración del Memory Copy para definir el registro a leer

Para separar las operaciones de lectura y escritura se utilizan unos comparadores, ya que cuando se genera la interrupción de lectura el número que genera el registro *I2CISRC* es 4 y cuando la interrupción generada es para

transmitir equivale al número 5. La configuración de los comparadores para cada caso se iguala al número según la interrupción correspondiente, de esta manera cuando se cumpla la condición la respuesta es una señal booleana. Esta señal activa los subsistemas tipo *Enable* en donde se encuentran los algoritmos correspondientes a la operación en curso, como se muestra en la Figura 3.22.

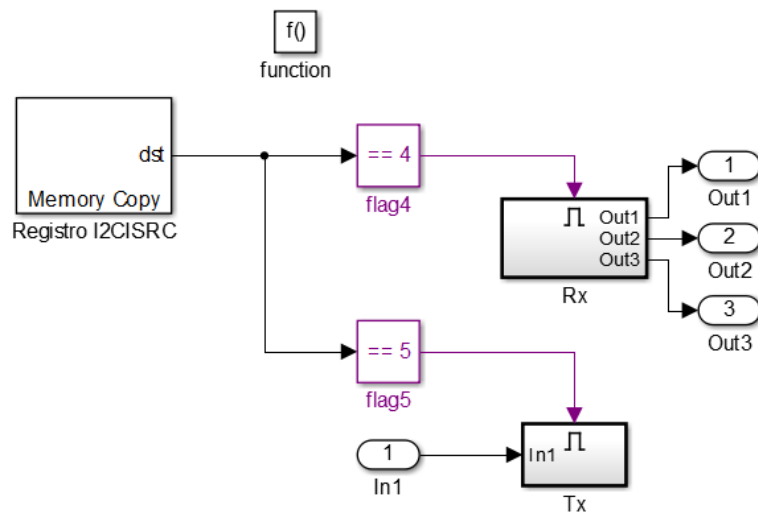


Figura 3.22 Código para escribir o leer según la interrupción que se activa

Dentro del subsistema llamado *Rx* se encuentra el algoritmo para recibir la información enviada por el Crius. Consta de un bloque *Receive I2C* configurado como lo muestra la siguiente imagen.

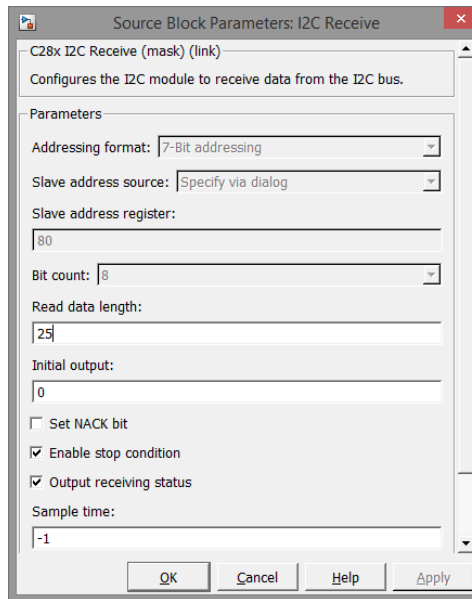


Figura 3.23 Configuración para leer por i2c

El motivo de que sean 25 datos es porque lo que recibe del Crius es: 3 ejes del acelerómetro y giroscopio, la guiñada del magnetómetro, los 4 canales del radiocontrol y las 3 ganancias de los controladores, es decir un total de 14 variables. Sin embargo como el bus del protocolo I2C sólo soporta números de 8 bits y todas las mediciones a excepción de las ganancias son de 16 bits, éstas se transmiten en dos partes. Por lo tanto, es necesario unir dichas mitades y además cambiar el tipo de dato int16 a double para todas las variables de medición incluyendo las ganancias.

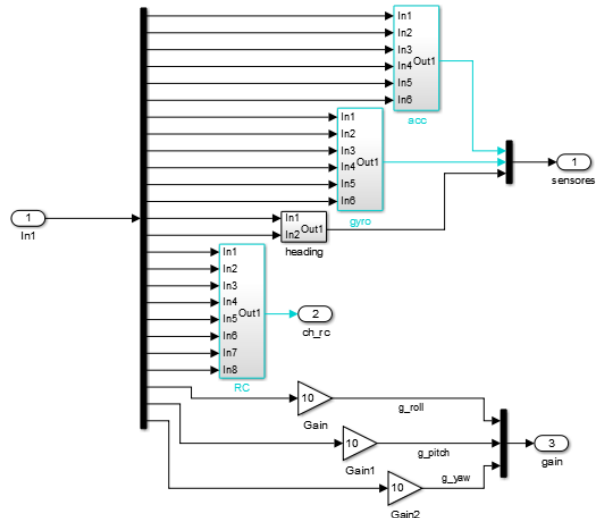


Figura 3.24 Código de unión de las variables que recibe el Piccolo del Crius

En el caso de las ganancias de los controladores, se agrega un multiplicador para aumentar o disminuir la sensibilidad porque en la aplicación sólo se puede cambiar de 0 hasta 20 con incremento de 0.1, sin embargo dentro del código del Crius estos valores se multiplican por 10 lo que significa que en realidad las ganancias cambian de 0 a 200.

Para obtener las posiciones angulares estimadas de los tres ejes del vehículo, alabeo, cabeceo y guiñada, se utiliza el acelerómetro, giroscopio y magnetómetro pero cada eje usa diferentes mediciones de los sensores. La configuración para cada eje se observa en la siguiente tabla:

Tabla 3-1 Ejes utilizados para la estimación de las posiciones angulares

Eje	Acelerómetro	Giroscopio	Magnetómetro
Alabeo (Roll)	Acc_roll, Acc_yaw	Gyro_roll	No aplica
Cabeceo (Pitch)	Acc_pitch, Acc_yaw	Gyro_pitch	No aplica
Guiñada (Yaw)	No aplica	Gyro_yaw	Heading

Los algoritmos de las estimaciones de alabeo y cabeceo siguen el mismo principio de los usados en el código del PVTOL. La posición angular estimada se

obtiene de un observador de Luenberger que tiene como entradas la velocidad angular del giroscopio y la posición angular obtenida del acelerómetro.

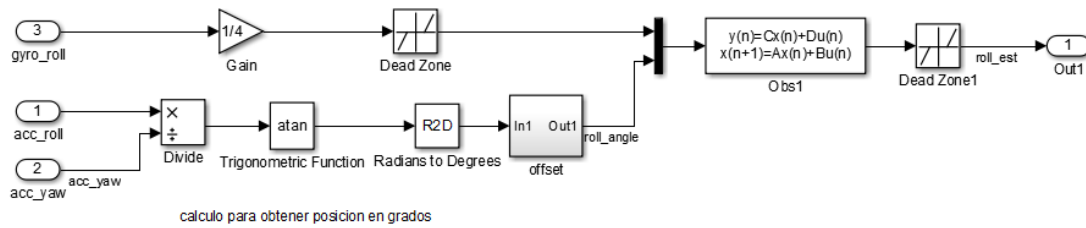


Figura 3.25 Código para estimar la posición angular de alabeo

La señal del giroscopio se multiplica por 0.25 para acortar la amplitud de la medición, además pasa por una zona muerta para mejorar la señal.

En el caso de las señales del acelerómetro se obtiene la posición angular con $\text{atan}(\text{acc_roll}/\text{acc_yaw})$ y las unidades se convierten de radianes a grados. Después se calcula y se elimina el offset de la posición angular, de esta manera se asegura que la posición angular sea cero en la posición inicial del vehículo.

La secuencia para calcular y eliminar el offset dura 10 segundos y consta de dos secciones, en la primera se calcula el promedio de la posición angular calculada a partir del acelerómetro y al pasar los 10 segundos el resultado se le resta a la posición angular, mientras que la segunda sección no deja pasar a la señal antes que pasen los 10 segundos y cuando pasa se encuentra una zona muerta para mejorar un poco más la medición.

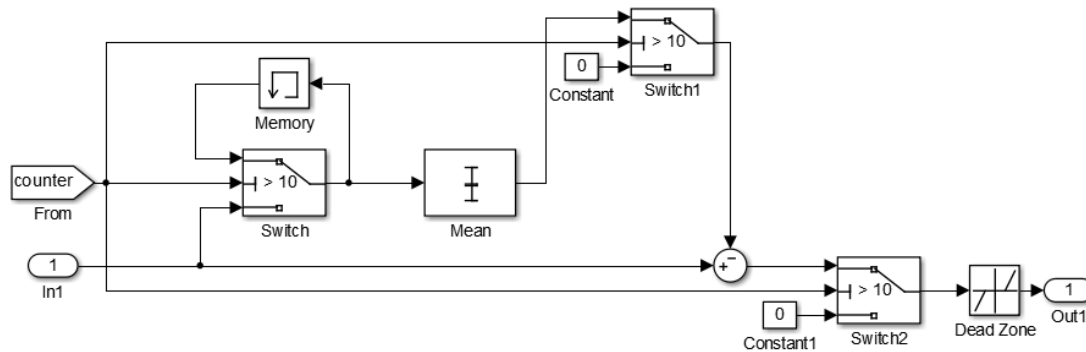


Figura 3.26 Código para calcular y eliminar el offset

Para el algoritmo de la estimación de guiñada sigue el mismo principio que el usado en los ejes anteriores pero con la diferencia de que para la posición angular se mide con el magnetómetro. El Crius envía la guiñada (heading) calculada con los ejes x y y del magnetómetro y tiene un rango de -180 a 180 grados. Sin embargo es necesario calcular y eliminar el offset para que independientemente de la orientación que tenga el cuadri-rotor al inicio con respecto al Norte sea cero.

Sólo dentro del subsistema que elimina el offset de la guiñada se incluye el led que sirve de guía visual para determinar cuando el sistema termina de eliminar el offset en los tres ejes. Recordando, el Piccolo cuenta con dos led uno de color verde que siempre está encendido y otro de color rojo que está conectado al puerto *GPIO34*, por lo tanto se define este puerto como salida digital y se conecta en la línea del resultado del cálculo del offset con la siguiente estructura. Primero la señal se convierte a positiva por medio del valor absoluto y después entra a un comparador con la condición mayor que cero, mientras se calcula el offset el led esta encendido y cuando se calcula el offset la condición se cumple y el led se apaga.

Por último es necesario acondicionar la señal de la guiñada ya que dicha señal recibida después de eliminar el offset pierde la escala original. Para corregirla se utiliza la función $mod(in,esc)$ que generalmente se utiliza para obtener el residuo de una división, donde in es el dividendo y esc el divisor; también se utiliza para sumar, restar o multiplicar dos números y ajustar la respuesta dentro de una escala determinada, como un reloj aritmético, por ejemplo: $mod(5+3,6) = mod(8,6) = 2$, porque la escala es de 0 a 5.

La lógica es la siguiente: si la señal es de 0 a 180 el divisor es 180, si la señal va de 0 a -180 el divisor es -180, cuando la señal sea mayor a 180 el divisor es -180 y por último cuando la señal sea menor que -180 el divisor es 180. El algoritmo empleado se muestra en la siguiente imagen, donde la entrada es la señal sin offset.

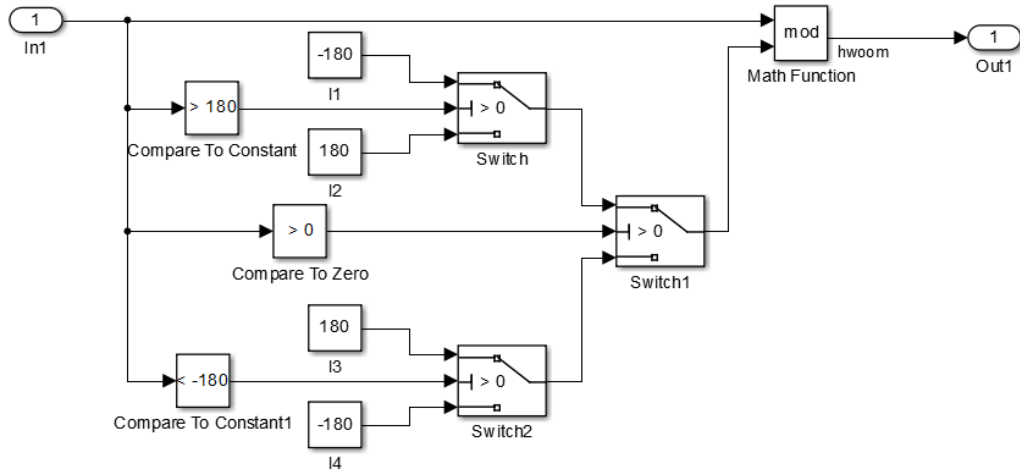


Figura 3.27 Código para escalar la señal de la guiñada de -180 a 180

3.7.3 Referencias del sistema

Las referencias del sistema son tres de los cuatro canales del radiocontrol, el canal 1, 2 y 4 para cabeceo, alabeo y guiñada respectivamente, el canal 3 es el acelerador de los motores.

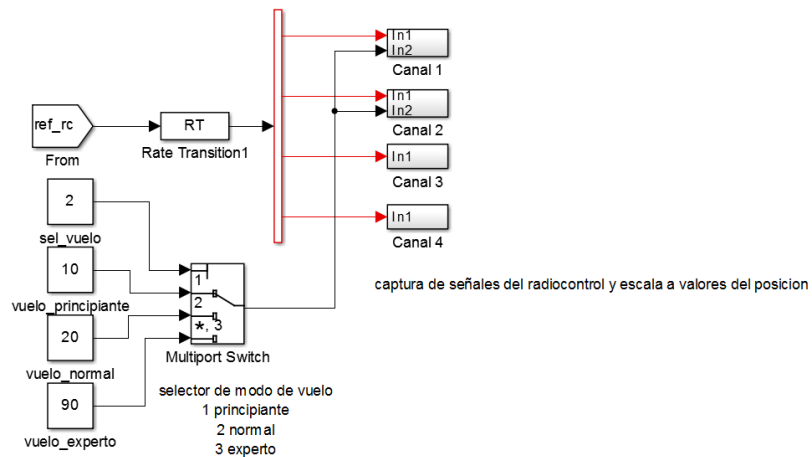


Figura 3.28 Código para seleccionar modo de vuelo de los canales 1 y 2

Para los canales 1 y 2 se agregó un apartado para modificar el rango de operación de las referencias y de esta manera cambiar el tipo de modo de vuelo. De este modo se permite ajustar los controles para alabeo y cabeceo según las habilidades del operador.

Los tres canales (1, 2 y 4) se escalan en las unidades que tienen los sensores, que son grados, ya que se mide la posición angular. Cada canal además cuenta con un filtro pasa bajo para suavizar los cambios bruscos tipo escalón de las referencias.

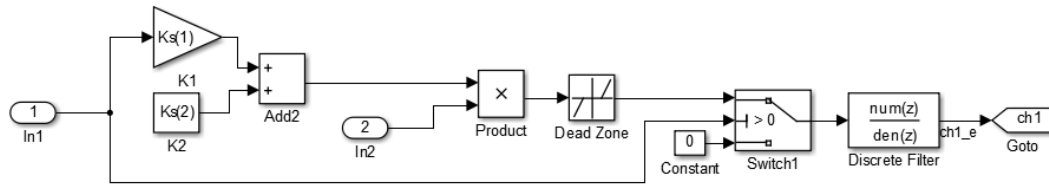


Figura 3.29 Código para escalar la referencia de los canales 1 y 2

A diferencia de los canales 1 y 2 donde la posición natural de las referencias es cero, debido al mecanismo del radiocontrol, en el canal 4 es necesario que se mantenga la posición de la referencia después de moverla. Para hacer esto se incluye un integrador para definir si el cambio es positivo o negativo y un ajuste sencillo con el operador módulo, con el cual si es mayor a cero el divisor es 180 y si es menor que cero el divisor es -180. De esta manera la referencia cambia de -180 a 180 en un rango pequeño. Además el integrador tiene que tener un retardo de 13 segundos al inicio ya que después de los 10 segundos donde se elimina el offset se cuenta con tres segundos para mover el canal 4 para activar los motores sin que se integre la señal manteniendo de esta manera la referencia de la guiñada en cero.

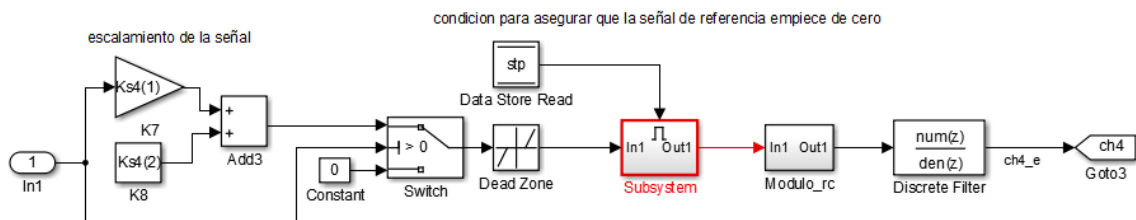


Figura 3.30 Código para escalar y mantener la referencia del canal 4

En el caso del canal tres se escala para 1142 hasta 1850, debido a que es el rango para que el valor mínimo no mueva los motores y el máximo no produzca la máxima capacidad de los motores.

3.7.4 Implementación de controladores

Para los controladores se utiliza una ganancia externa que se puede modificar desde la interfaz gráfica de la estación terrestre. Sin embargo, existe un apartado el cual mantiene las entradas de los controladores en cero para evitar que generen una deriva, la entrada se mantiene en cero por 13 segundos porque en los primeros 10 segundos el Piccolo calcula y elimina el offset, además para activar los motores se necesita mover el canal 4 del radiocontrol que al moverlo se modifica la referencia. Por lo tanto, se cuenta con tres segundos para activar los motores sin que se afecte la referencia de la guiñada y asegurar que se inicie en cero.

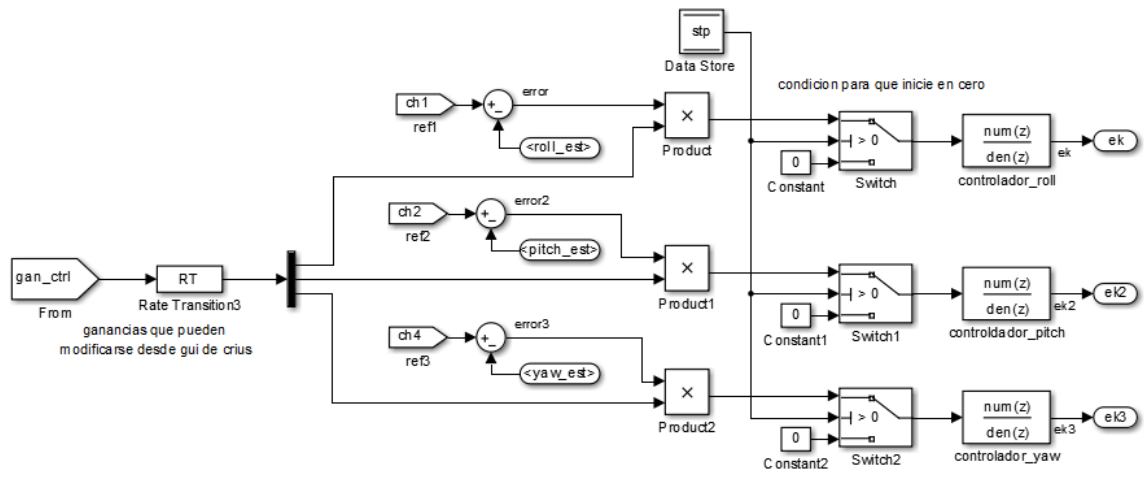


Figura 3.31 Código de la implementación de los controladores

3.7.5 Implementación de la ganancia dinámica y matriz de propulsión

Después de los controladores se encuentra el algoritmo para generar el valor que necesita el Crius para generar la señal PWM. Las primeras operaciones que se realizan son para ajustar la ganancia de forma dinámica para las entradas de

control del alabeo y cabeceo, este ajuste es para disminuir la ganancia conforme se aumenta la velocidad de los motores. La entrada de control de la guiñada así como el canal 3 del radiocontrol no sufren algún cambio.

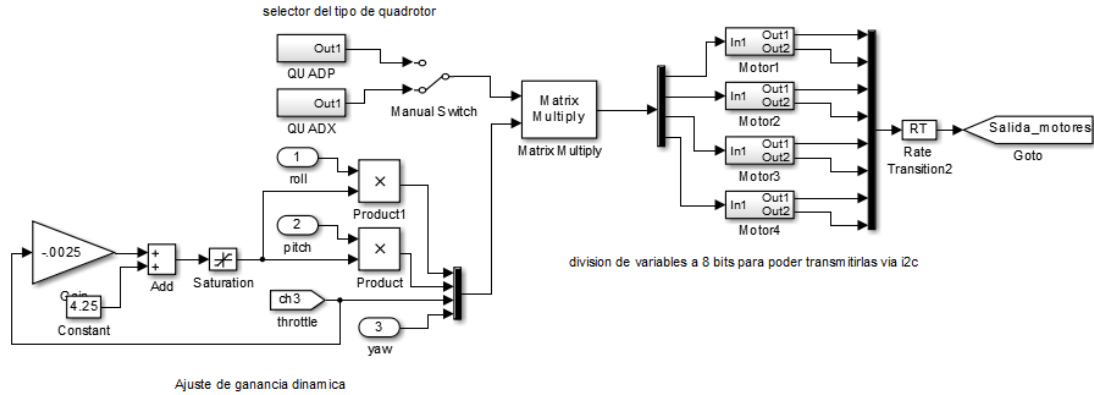


Figura 3.32 Código para generar los valores necesarios que el Crius necesita para construir el PWM

Después las tres entradas de control y el canal 3 se multiplican por la matriz de propulsión generada a partir de la configuración del cuadri-rotor ya sea “+” o “x”. La matriz es de 4x4 y se define según la configuración de los motores en el vehículo.

Al resultado se le añade un filtro pasa bajas para eliminar el ruido de alta frecuencia generado por las vibraciones de los motores y un bloque de saturación para limitar la señal con el valor mínimo y máximo que soporta el Crius para generar la señal PWM.

Como la resolución es de 16 bits se tiene que dividir en 2 partes de 8 bits para poder transmitirlos por el protocolo I2C. Primero se extraen los 8 bits más significativos y se convierte en entero sin signo de 8 bits (uint8), después se extraen los 8 bits menos significativos y se convierte a uint8; esta operación se realiza para las cuatro señales.

3.7.6 Escritura por I2C

Regresando al subsistema de la interrupción se agrega el apartado para escribir similar al generado para la lectura del I2C pero con la diferencia de que en el comparador la bandera para escribir es 5 y dentro del subsistema el bloque es *I2C Transmit*.

Existe una limitante de datos a transmitir de cuatro palabras de 8 bits, por lo que se genera una pequeña secuencia que permite transmitir en dos grupos las ocho palabras a transmitir.

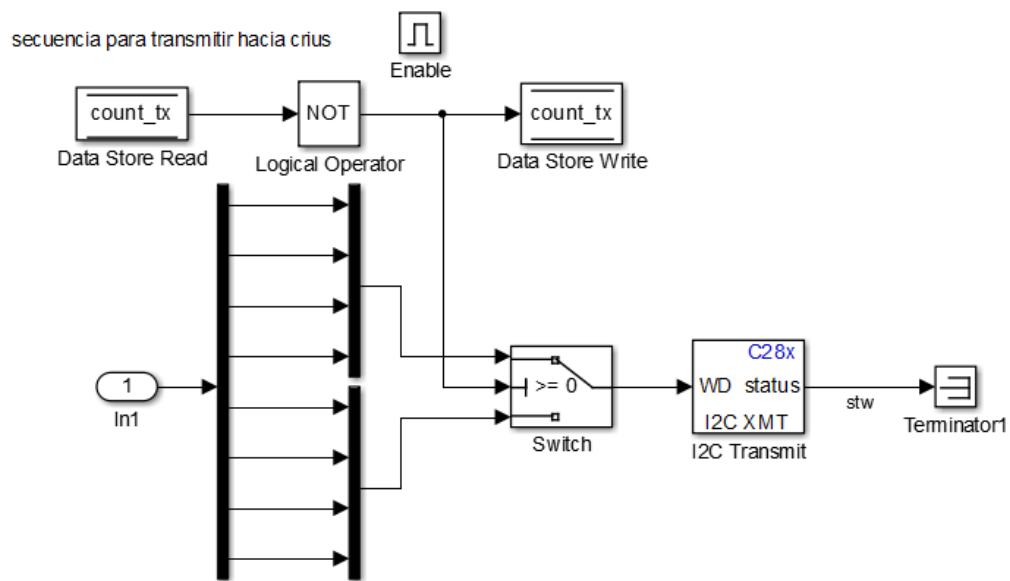


Figura 3.33 Código para transmitir por i2c las ocho mitades de las señales para PWM

3.8 Código para definir las variables a utilizar

Este código está en un script de Matlab el cual se tiene que ejecutar antes de compilar el código hecho en Simulink. El código del script se encuentra dentro de los anexos de esta tesis.

3.9 Código MultiWii 2.2

El código que utiliza el Crius es el *MultiWii 2.2* el cual se tuvo que modificar para satisfacer los requerimientos de la interacción entre el Crius y el Piccolo.

Este código se encuentra dividido en 1 programa principal, 9 subprogramas (cada uno realiza una función principal del sistema) y 3 librerías para la configuración general del sistema, esto hace un poco más accesible modificar el código.

Tabla 3-2 Descripción general del código MultiWii 2.2

Pestaña	Descripción
MultiWii	Código principal en donde se definen todas las variables globales, los pines del MCU, ejecuta el sistema general
Alarms	Código para activar las alarmas en caso de que la batería tenga bajo voltaje.
EEPROM	Memoria donde se almacenan los valores de las ganancias de los controladores entre otras variables más.
GPS	Código para configurar y tomar lectura del GPS. Se necesita un módulo GPS.
IMU	Código que estima las posiciones a partir de las lecturas de los sensores.
LCD	Contiene el código para la telemetría con una pantalla LCD.
Output	Código donde genera las señales PWM.
RX	Toma lectura de los puertos conectados al receptor del radiocontrol.
Sensors	Base de datos para configurar y tomar mediciones de diversos modelos de sensores comerciales (acelerómetro, giroscopio, magnetómetro, barómetro)
Serial	Apartado para comunicarse por medio de Serial con algún módulo Rc, GPS o Bluetooth
config.h	Librería para la configuración general del Crius.
def.h	Librería que contiene la configuración de los puertos según el MCU que utiliza la tarjeta, además define los sensores y su orientación según el modelo de la tarjeta utilizada.
tinygps.h	Librería para un módulo de GPS que se conecta por medio del I2C.

Lo primero que se modifica en el código es la librería *config.h* aquí se define el tipo de vehículo que en este caso es un cuadri-rotor tipo x, los valores mínimo y máximo que pueden tomar las salidas para las señales PWM, la velocidad del reloj del I2C (400 kHz), el modelo de la tarjeta que se está usando que es Crius_SE, entre otras opciones que se pueden modificar.

En la librería *def.h* se modificó el apartado del CRIUS_SE para agregar los sensores con los que cuenta el modelo del Crius.

```
#if defined(CRIUS_SE)
#define MPU6050
//#define ITG3200
//#define BMA180
#define HMC5883
#define BMP085
#define ACC_ORIENTATION(X, Y, Z) {accADC[ROLL] = -X; accADC[PITCH] = -Y; accADC[YAW] = Z;}
#define GYRO_ORIENTATION(X, Y, Z) {gyroADC[ROLL] = Y; gyroADC[PITCH] = -X; gyroADC[YAW] = -Z;}
#define MAG_ORIENTATION(X, Y, Z) {magADC[ROLL] = X; magADC[PITCH] = Y; magADC[YAW] = -Z;}
#endif
```

Figura 3.34 Cambios en el código para definir los sensores disponibles en el Crius

Los cambios más significativos que tiene el código principal *MultiWii* se encuentran al final, donde el apartado de control se comenta y de esta manera el Crius ya no puede realizar los algoritmos de control. En su lugar se agrega el apartado relacionado con la comunicación I2C que va a tener con el Piccolo, primero se escriben las variables a transmitir y después se transmiten. Cada variable se divide en 2 por el tamaño que tiene, con excepción de las ganancias que se transmiten sin dividir siendo en total 25 datos a transmitir.

```

MultiWii  Alarms  EEPROM  GPS  IMU  LCD  Output  RX  Sensors  Serial  config.h  de
GPS_angle[PITCH] = (nav_rated[LON]*sin_yaw_y + nav_rated[LAT]*cos_yaw_x) /10;
#else
GPS_angle[ROLL] = (nav[LON]*cos_yaw_x - nav[LAT]*sin_yaw_y) /10;
GPS_angle[PITCH] = (nav[LON]*sin_yaw_y + nav[LAT]*cos_yaw_x) /10;
#endif
} else {
GPS_angle[ROLL] = 0;
GPS_angle[PITCH] = 0;
}
#endif
/*
/**** PITCH & ROLL & YAW PID ****
int16_t prop;
prop = min(max(abs(rcCommand[PITCH]),abs(rcCommand[ROLL])),500); // range [0;500]

for(axis=0;axis<3;axis++) {
if ((f.ANGLE_MODE || f.HORIZON_MODE) && axis<2) { // MODE relying on ACC
// 50 degrees max inclination
errorAngle = constrain((rcCommand[axis]<<1) + GPS_angle[axis],-500,+500) - angle[axis] + conf.
PTermACC = ((int32_t)errorAngle*conf.P8[PIDLEVEL])>>7; // 32 bits is
PTermACC = constrain(PTermACC,-conf.D8[PIDLEVEL]*5,+conf.D8[PIDLEVEL]*5);
}
}
}

```

Figura 3.35 Código comentado de la sección de los controladores

```

MultiWii  Alarms  EEPROM  GPS  IMU  LCD  Output  RX  Sensors  Serial  config.h  de
//transmission hacia piccolo por i2c
unsigned int vec[25]={highByte(angle[ROLL]),
                    lowByte(angle[ROLL]),
                    highByte(angle[PITCH]),
                    lowByte(angle[PITCH]),
                    highByte(accADC[YAW]),
                    lowByte(accADC[YAW]),
                    highByte(gyroADC[ROLL]),
                    lowByte(gyroADC[ROLL]),
                    highByte(gyroADC[PITCH]),
                    lowByte(gyroADC[PITCH]),
                    highByte(gyroADC[YAW]),
                    lowByte(gyroADC[YAW]),
                    highByte(heading),
                    lowByte(heading),
                    highByte(rcData[ROLL]),
                    lowByte(rcData[ROLL]),
                    highByte(rcData[PITCH]),
                    lowByte(rcData[PITCH]),
                    highByte(rcData[THROTTLE]),
                    lowByte(rcData[THROTTLE]),
                    highByte(rcData[YAW]),
                    lowByte(rcData[YAW]),
                    conf.P8[ROLL],
                    conf.P8[PITCH],
                    conf.P8[YAW]};

```

Figura 3.36 Código añadido para definir las variables a transmitir por i2c

Para la toma de lectura del Piccolo se agregaron dos subrutinas las cuales se llaman después de que se realiza la transmisión de las variables. Cada subrutina sirve para leer a uno de los dos grupos de variable que transmite el Piccolo.

```
MultiWii  Alarms  EEPROM  GPS  IMU  LCD  Output  RX  Sensors  Serial  config.h  de
//Wire.beginTransmission(10); // transmitir hacia piccolo
i2c_rep_start(10<<1);
for (i=0;i<25;i++){
i2c_write(vec[i]);
i2c_stop();
//lectura de datos desde piccolo
//Wire.requestFrom(10,2);
//uint16_t ytse=Wire.read()<<8;
//uint8_t ytse1=Wire.read();
//delay(1);
i2c_read_piccolo_f4();
i2c_read_piccolo_s4();
```

Figura 3.37 Código agregado para transmitir y recibir las variables según sea el caso

Las subrutinas se agregaron en el subprograma “Sensors” como se aprecia en la Figura 3.38, en el que además de la base de datos de los sensores, se definen las subrutinas para las instrucciones del I2C.

```
MultiWii  Alarms  EEPROM  GPS  IMU  LCD  Output  RX  Sensors  Serial
uint8_t i2c_readReg(uint8_t add, uint8_t reg) {
    uint8_t val;
    i2c_read_reg_to_buf(add, reg, &val, 1);
    return val;
}

void i2c_read_piccolo_f4(){
    i2c_read_to_buf(10, rawpiccolo, 4);
    i2c_stop();
    motor[3]=rawpiccolo[0]<<8|rawpiccolo[1];
    motor[1]=rawpiccolo[2]<<8|rawpiccolo[3];
}

void i2c_read_piccolo_s4(){
    i2c_read_to_buf(10, rawpiccolo, 4);
    i2c_stop();
    motor[0]=rawpiccolo[0]<<8|rawpiccolo[1];
    motor[2]=rawpiccolo[2]<<8|rawpiccolo[3];
}

// *****
// GYRO common part
// *****
```

Figura 3.38 Código de las subrutinas para tomar lectura de la información enviada por el Piccolo

3.10 Resultados experimentales

Utilizando los bancos de pruebas de las secciones 3.2.1 y 3.2.2 se realizaron diferentes experimentos para ajustar y verificar el funcionamiento del sistema final antes de los últimos experimentos que constan del vuelo del vehículo. Sin embargo, de estos experimentos los resultados que tienen relevancia son los realizados en el banco que tiene la rótula ya que se puede apreciar el comportamiento del cuadri-rotor en los tres ejes.

Los resultados se muestran en las siguientes figuras, como se puede ver se realizaron con movimientos aislados para poder observar los cambios en los otros ejes cuando se presenta un cambio en uno de ellos; es decir, el acoplamiento. En las siguientes figuras, la línea azul representa la referencia que se mandó por medio de un control remoto, mientras que la línea verde representa la salida observada.

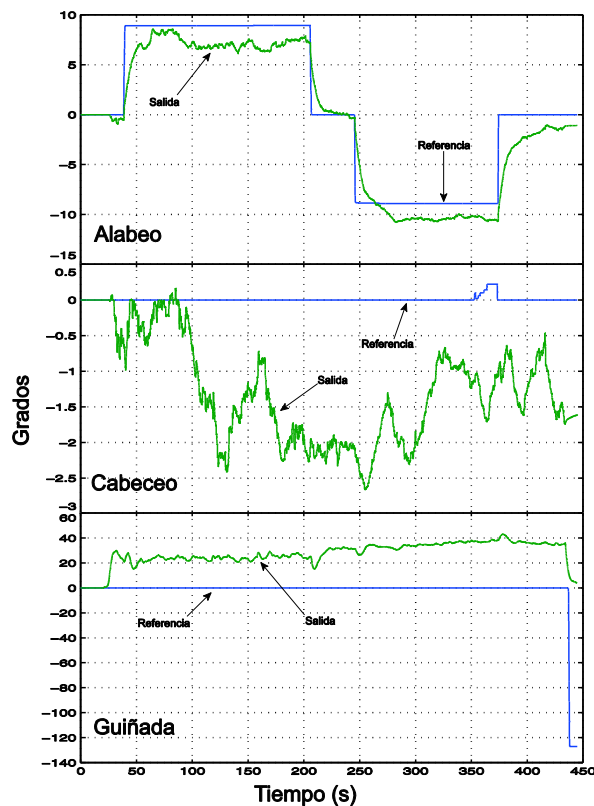


Figura 3.39 Respuesta experimental modificando sólo la referencia de alabeo

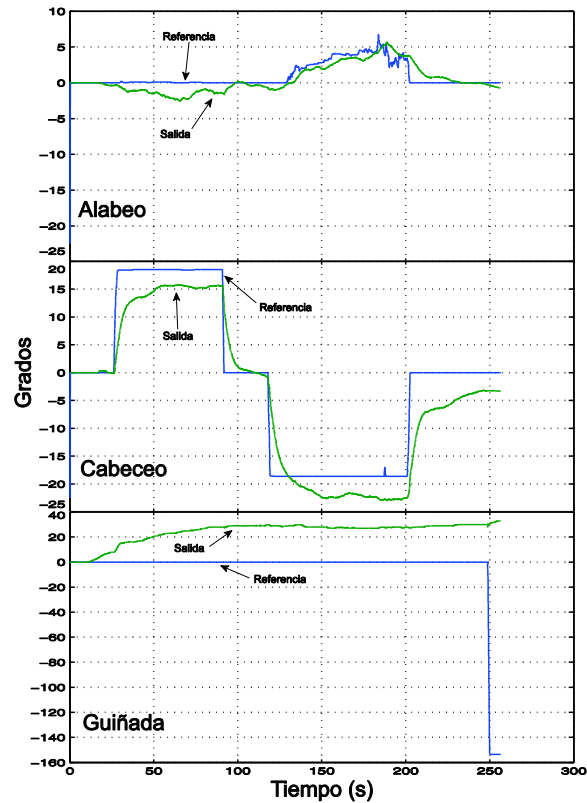


Figura 3.40 Respuesta experimental modificando la referencia de cabeceo

El error presente en los ejes de cabeceo y alabeo está en un rango de ± 2 grados lo cual es aceptable considerando que el banco de pruebas tiene fricción estática en la rótula, misma que no está presente en el modelo matemático y tiene un efecto importante en la dinámica del sistema.

Por otro lado, se puede notar en las Figura 3.39 y la Figura 3.40 que en la guiñada existe un error constante. Éste se debe a que el conjunto de los motores con las hélices no son iguales sino que tienen diferencias; es decir, no tienen la misma velocidad angular. Este efecto se debe tanto a los motores mismos como a la electrónica de potencia y a las mismas hélices. Por lo tanto, el vehículo tiene una tendencia a rotar en ese eje en lazo abierto, de tal manera que en lazo cerrado este desbalance se rechaza hasta que el nivel de error es alto y la fuerza de control se equilibra con el desbalance.

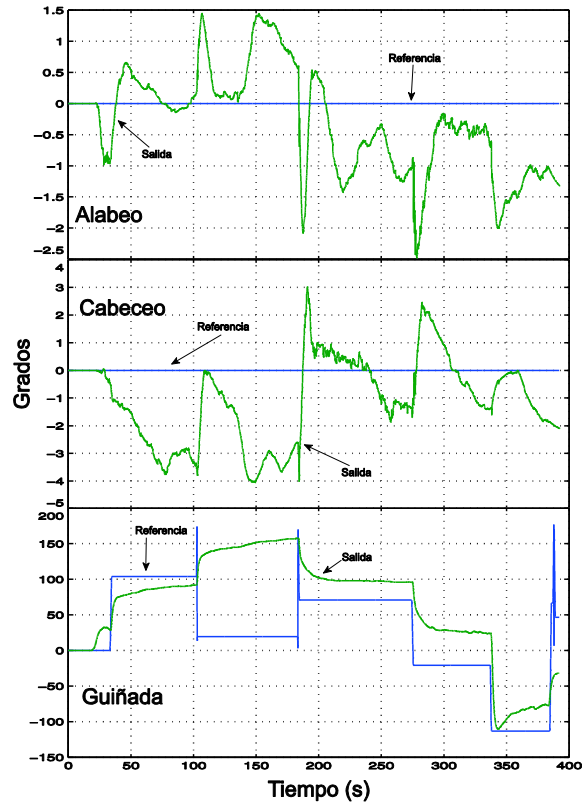


Figura 3.41 Respuesta experimental modificando la referencia de la guiñada

El error en la guiñada también se encuentra presente en la Figura 3.41; durante los experimentos el error en este eje no se afecta en gran medida la operación del vehículo ya que el piloto tiende a ajustar las referencias de los ejes de manera manual (“trim”). También sería posible compensar internamente este desbalance agregando una constante a la referencia del controlador de guiñada.

Hay que aclarar que para los experimentos en el banco de pruebas con la rótula, después de determinar los controladores a utilizar, sólo se modificaron las ganancias de los controladores con el fin de contrarrestar la fricción generada por la rótula, ya que en el aire, esta fricción no está presente. Estas ganancias fueron altas en comparación a las usadas en el vuelo libre del cuadri-rotor.

Antes de comenzar los experimentos de vuelo libre tuvo lugar una serie de pruebas en las cuales se suspendió el cuadri-rotor por la parte superior con un

hilo a un tubo para poder observar el comportamiento del vehículo con las ganancias configuradas durante los experimentos en el banco con la rótula. Como las ganancias fueron altas el cuadri-rotor comenzó a oscilar, por lo tanto el sistema era inestable; por eso se optó realizar estas pruebas preliminares al vuelo libre.

Como ya se ha mencionado las CDV comerciales utilizan controladores PID, de esta manera existe un método sencillo para ajustar las ganancias de dichos controladores. Este método consiste en modificar una ganancia a la vez, comenzando por la ganancia proporcional, seguida de la ganancia integral y por último la ganancia derivativa. Dicha modificación consiste en ir incrementando poco a poco la ganancia en curso hasta que el cuadri-rotor comience a oscilar ligeramente. Al notar la oscilación el siguiente paso es disminuir un poco hasta que deje de oscilar y esta ganancia es la que se deja finalmente.

Por lo tanto, tomando en cuenta el método descrito anteriormente las ganancias se ajustaron de forma similar, sólo que en este caso se modifican tres ganancias una para cada controlador. Como el controlador de la guiñada se ajustó cuando el cuadri-rotor se montó en el banco de pruebas de la sección 3.2.1, esta ganancia no tuvo muchos cambios. Sin embargo, para los otros dos ejes se tuvo que bajar las ganancias hasta en un 60% (con respecto a las ganancias utilizadas en el banco de la rótula) hasta que el vehículo dejó de tener oscilaciones. Los resultados de los experimentos en vuelo libre se muestran a continuación.

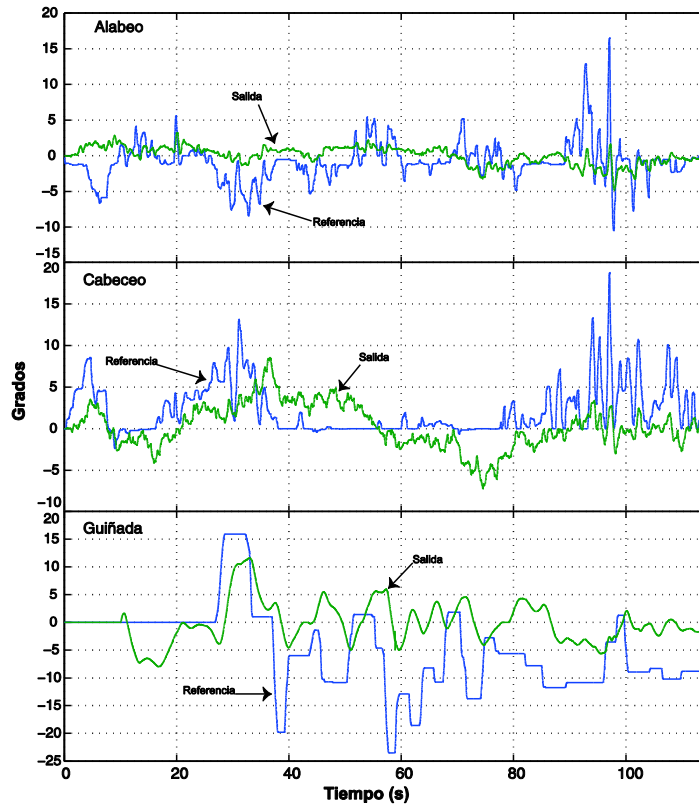


Figura 3.42 Resultados experimentales del cuadri-rotor en vuelo libre

Como las pruebas en vuelo libre fueron en interiores, los motores no están balanceados y la posición del sistema no es estable en lazo abierto aun cuando la orientación del mismo si lo sea, cualquier perturbación tiende a mover la posición del vehículo. Para mantenerlo en un rango de posición y evitar alguna colisión del vehículo, es necesario mover las referencias de la orientación constantemente. Sin embargo y a pesar del diseño del observador para estimar las posiciones angulares, el sistema presenta un buen seguimiento a la referencia con un error aceptable y fue posible volarlo fácilmente mediante gracias a que el vehículo es estable en orientación.

4 Conclusiones

Dados los resultados experimentales obtenidos se puede concluir que si es posible complementar una CDV comercial de bajo costo con un DSP con mejor desempeño. En particular, esto permitió definir un proceso de operación con prestaciones apropiadas para el desarrollo de controladores experimentales.

La Figura 4.1 muestra el proceso de la implementación y operación final que tiene la CDV Crius junto al DSP Piccolo, con el cual cumple la función de ser sencillo el implementar controladores para cuadri-rotors. Las secciones que se encuentran dentro de las líneas punteadas consisten en el trabajo principal de esta tesis. El resto del diagrama de flujo corresponde a la implementación de los controladores en el Piccolo. La razón por la cual se eligió programar al Piccolo por medio de Simulink-Matlab es que cualquier investigador o estudiante que tenga el interés de diseñar sus controladores a partir de un modelo matemático de un cuadri-rotor, pero que no tenga el conocimiento para implementar dichos controladores en un lenguaje de programación típico de los microcontroladores como es el C, C++, etc., pueda tener esta alternativa de implementar sus controladores de una forma más sencilla y casi directa de como los diseñan.

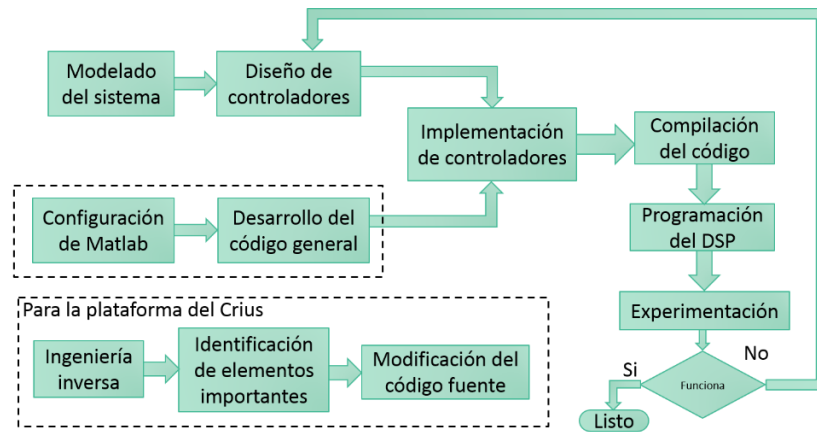


Figura 4.1 Diagrama de flujo del proceso de diseño/rediseño de controladores

Al implementar los controladores durante la experimentación es necesario modificar las ganancias debido a que no siempre funciona el controlador teórico la primera vez que se implementa. Si al modificar las ganancias no se obtiene un resultado positivo, entonces es cuando hay que rediseñar dichos controladores tomando en cuenta lo observado durante los experimentos previos. De esta manera se adquiere el aprendizaje del control aplicado.

Aunque la implementación de los controladores en el Piccolo usando Simulink es sencilla, no lo es así al momento de realizar las configuraciones necesarias para operar el control de flujo, comunicaciones, inicialización, elementos de seguridad, interrupciones y otros procesos que no están directamente relacionados con el esquema de control. Tal es el caso de las interrupciones usadas en la sección 3.7.2, las cuales para activarlas no se encuentran dentro del menú gráfico de Simulink, y se tuvieron que activar de forma manual con ayuda del Stateflow.

De los experimentos del PVTOL dentro del túnel de viento se puede concluir que los efectos de la ráfaga de viento afectan mucho al sistema, por lo que es necesario que el sistema cuente con un alto rechazo a perturbaciones de entrada. Este es un punto importante debido a que de eso depende el funcionamiento del vehículo en vuelo libre. Por lo tanto, se debe de considerar este tipo de

perturbaciones como un sistema complejo y no como típicamente se toman en cuenta las perturbaciones en un sistema.

De la CDV Crius fue posible sacar provecho de los elementos con los que cuenta tales como:

1. Interfaz gráfica de estación terrestre.
2. Captura de las señales del radiocontrol.
3. Conectores de los motores.
4. Comunicación por medio de Bluetooth.
5. Adquisición de las mediciones de los sensores.
6. Arquitectura abierta

Así fue posible extender las capacidades de la CDV con otro DSP, teniendo como resultado una CDV con potencial de desarrollo.

Por otra parte, se cumplieron la mayoría de los objetivos planteados en esta tesis. A excepción de la implementación de un controlador para la altitud. Este controlador no se implementó porque al estar en etapa experimental los vuelos serían a baja altitud. El sensor que se tenía pensado utilizar era el barómetro, sin embargo, este sensor tiene mal desempeño a baja altitud razón por la cual se tomó la decisión de no utilizarlo en algún lazo de control.

Los elementos de propulsión al no estar balanceados afectan al comportamiento que tiene el cuadri-rotor. Este efecto se ve reflejado principalmente en el eje correspondiente a la guiñada de las figuras 3.39 y 3.40.

El uso de plataformas experimentales para realizar pruebas del vehículo ayuda mucho en este tipo de sistemas porque, a pesar de incluir la fricción estática al sistema (puede considerarse como perturbación), reducen el tiempo de la implementación de controladores, es más apreciable la respuesta que tiene el sistema con los controladores, como el cuadri-rotor está sujeto a la base brinda un poco más seguridad para evitar accidentes.

4.1 Trabajo a futuro

Aunque el sistema desarrollado en esta tesis es funcional, los algoritmos del piloto automático son muy básicos, es decir, sólo controlan las posiciones angulares, alabeo, cabeceo y guiñada. Por lo cual se propone mejorar estos algoritmos tomando en cuenta las siguientes opciones:

- Incluir en los algoritmos de navegación los ejes traslacionales x , y , z .
- Mejorar la navegación implementando un GPS.

También implementar el controlador para la altitud y de esta forma el cuadri-rotor podría volar a mayor altitud. De esta manera la CDV adquiere más autonomía.

Caracterizar los motores para que generen la misma fuerza de empuje, ajustar la matriz de propulsión y así evitar que el sistema tenga errores como los observados en el eje de la guiñada de la Figura 3.42.

4.2 Recomendaciones

Se recomienda balancear bien las hélices antes de volar el cuadri-rotor para evitar un mayor desbalanceo.

Al armar la estructura y montar los motores, se aconseja utilizar un pegamento para roscas removible, el cual se le aplica a los tornillos para que no se aflojen con las vibraciones, por ejemplo el Loctite 242.

Hay que tener cuidado con la batería que se utiliza, generalmente son tipo Li-Po. Este tipo de baterías tiene la peculiaridad de que cada celda tiene un voltaje nominal de 3.7v, por lo tanto, según la cantidad de celdas que tenga se multiplica por el voltaje nominal, por ejemplo, en esta tesis se usó una batería de 3 celdas por lo que el voltaje nominal es de 11.1v. Pero cuando se carga al máximo, cada celda tiene 4.2v por lo que el voltaje máximo de la batería es de 12.6v. Sin embargo, donde hay que prestar más atención es en el voltaje mínimo que por

cada celda es de 3v. Esta es una característica importante debido a que uno de los motivos por lo cual las baterías dejan de funcionar, es porque se descargan más del voltaje mínimo y es muy difícil recuperar la carga. Esto sucede ya que comúnmente los cargadores de baterías cuentan con una protección que les impiden cargar baterías que excedan el voltaje mínimo.

Por eso cuando se utilice una batería en el cuadri-rotor, hay que tener cuidado con el tiempo de uso. Si se observa que el sistema de propulsión baja el empuje generado es que se sobrepasó el voltaje nominal de la batería. Se recomienda dejar de utilizar la batería cuando alcance su voltaje nominal. Aunque si el voltaje se encuentra por debajo, los cargadores aún la pueden cargar pero no se recomienda hacerlo debido a que se recorta su tiempo de vida.

Hay que hacer la aclaración de que en las baterías el voltaje no es lo más importante, no obstante es la variable con la que es más fácil explicar el funcionamiento. La variable de la cual se tiene que tener más en cuenta es el valor de la descarga, éste número comúnmente se encuentra en las baterías con una letra C al final, por ejemplo 25C. Por último se recomienda buscar más información acerca de la descarga.

5 Bibliografía

- [1] R. Austin, Unmanned aircraft systems: UAVS design, development and deployment, Wiley, 2010.
- [2] T. W. M. Randal W. Beard, Small Unmanned Aircraft: Theory and Practice, Princeton University Press, 2012.
- [3] M. K. B. A. A. Mustafa Ilarslan, «Avionics System Design of a Mini VTOL UAV,» *29th Digital Avionics Systems Conference*, 2010.
- [4] J. J. O. R. T. C. Y. B. M. C. T. H. L. Swee King Phang, «Autonomus Mini.UAV for Indoor Flight with Embedded On-board Vision Prossesing as Navigation System,» *IEEE Region 8 SIBRCON*, 2010.
- [5] J. L. P. R. Enric Pastor, «UAV Payload and Mission Control Hardware/Software Architecture,» *IEEE A&E Systems Magazine*, 2007.
- [6] Y. L. Yi- Rui Tang, «The Software of a Reconfigurable Real-Time Onboard Control System for a Small UAV Helicopter,» *8th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, 2011.
- [7] Y. C. Y. Haiyang Chao, «Autopilots for Small Fixed-Wing Unmanned Air Vehicles: A Survey,» *International Conference on Mechatronics and Automation*, 2007.
- [8] S. S. G. M. J. Hauser, «Nonlinear control design for slightly non-minimum phase systems: Application to V/STOL aircraft,» *Automatica*, vol. 28(4), pp. 665-679, 1992.
- [9] E. A.-B. V. Rejón, «Discrete-time stabilization of a PVTOL without roll angle and velocities measurement,» *45th Conference on Decision & Control*, 13-15 December 2006.
- [10] P. E. Dupont, «Avoiding stick-slip through PD control,» *IEEE Transactions on Automatic Control* , vol. 39, nº 5, pp. 1094-1097, 1994.
- [11] A. M.-T. M. T.-F. D. & T. K. Kenfack-Jiotsa, «Dry friction: motion - map, characterization and control,» *European Physical Journal*, 2012.

- [12] L. A.-B. J. L.-C. E. L.-C. D. Hernández-Alcantara, «Control conmutado para un sistema de levitación magnética con atascamiento-deslizamiento,» *Revista Iberoamericana de Automática e Informática Industrial RIAI*, Vols. %1 de %211-3, pp. 285-294.
- [13] T. Instruments, «TMS320x2806x Technical Reference Manual,» Marzo 2016.
- [14] L. A.-B. E. L.-C. M. González-Sánchez, «Simplificación de Controladores para Cuadrirotos Utilizando Modelos de Diseño Simplificados,» *AMCA*, 2013.
- [15] Multiwii, «www.multiwii.com,» [En línea].
- [16] J. A. Farrell, *Aided Navigation: GPS with High Rate Sensors*, McGraw-Hill, 2008.

Índice de figuras

Figura 1.1 Arquitectura del sistema de una CDV autónoma [2]	15
Figura 2.1 Modelo del PVTOL (Planar Vertical Takeoff and Landing)	25
Figura 2.2 Banco de pruebas.....	27
Figura 2.3 Comparación entre la respuesta en lazo cerrado experimental y el modelo	29
Figura 2.4 Diagrama esquemático	30
Figura 2.5 Placa de conexiones PCB	31
Figura 2.6 Observador basado en la mezcla de sensores.....	33
Figura 2.7 Respuesta en frecuencia del observador considerando la contribución del giroscopio y la contribución del acelerómetro	34
Figura 2.8 Respuesta en frecuencia de lazo abierto del ángulo de alabeo considerando el controlador (14)	36
Figura 2.9 Características del rechazo a perturbaciones.....	37
Figura 2.10 Código general del PVTOL	38
Figura 2.11 Bloques de recepción y transmisión por I2C.....	39
Figura 2.12 Configuración básica para transmitir un dato por medio de I2C	40
Figura 2.13 Configuración para activar la interrupción.....	42
Figura 2.14 Primera parte de la comunicación para leer cualquier registro por I2C	42
Figura 2.15 Configuración del bloque Hardware Interrupt	43
Figura 2.16 Subsistema para ejecutar código con interrupción	44
Figura 2.17 Configuración del bloque Memory Copy	44
Figura 2.18 Código para determinar el sensor a leer.....	45
Figura 2.19 Código y configuración para leer datos por I2C.....	46
Figura 2.20 Código para concatenar la medición.....	47
Figura 2.21 Bloque para leer por I2C con la bandera estatus.....	48
Figura 2.22 Librería de Stateflow	49
Figura 2.23 Definición de variables y funciones de Stateflow	50
Figura 2.24 Flujo de los estados del Stateflow.....	51

Figura 2.25 Bloque Stateflow con las funciones de salida	53
Figura 2.26 Condiciones de entrada al Stateflow.....	54
Figura 2.27 Código para estimar la posición angular de alabeo	55
Figura 2.28 Código para seleccionar el tipo de referencia.....	55
Figura 2.29 Bloque eCAP para capturar la señal del radiocontrol	56
Figura 2.30 Configuración del bloque eCAP	57
Figura 2.31 Código de adquisición de la señal del radiocontrol	58
Figura 2.32 Código para seleccionar el tipo de referencia interna	59
Figura 2.33 Código para el controlador.....	60
Figura 2.34 Código para generar las señales PWM	61
Figura 2.35 Configuración general del bloque ePWM.....	62
Figura 2.36 Configuraciones de las pestañas ePWMA y ePWMB del bloque ePWM.....	63
Figura 2.37 Variable a monitorear en el CCS	65
Figura 2.38 Configuración de la variable a monitorear	66
Figura 2.39 Pestaña Expressions con la variable agregada	66
Figura 2.40 Configuración del tiempo de muestreo del CCS	67
Figura 2.41 Configuración para graficar una variable	68
Figura 2.42 Código para seleccionar la fuente de la referencia	69
Figura 2.43 Respuesta experimental a diferentes velocidades en el túnel de viento usando escalones de referencia.....	70
Figura 2.44 Respuesta experimental a diferentes velocidades dentro del túnel de viento usando la señal Chirp como referencia	71
Figura 2.45 Esfuerzo de control normalizado en los experimentos de la Figura 2.43.....	73
Figura 3.1 Marco de referencia del cuadri-rotor y configuración de las hélices [14]	75
Figura 3.2 Ensamble de la base con movimiento en guiñada.....	79
Figura 3.3 Ensamble de la base con movimiento en los 3 ejes	79
Figura 3.4 Comparación de la salida observada que tiene el sistema al colocar el polo del observador en -0.1, -0.2 y -0.4.	81

Figura 3.5 Respuesta en frecuencia del sistema en lazo abierto para los controladores C1 y C2	83
Figura 3.6 Respuesta en el tiempo del sistema en lazo cerrado ante una entrada escalón para cada controlador.....	83
Figura 3.7 Respuesta en frecuencia del sistema en lazo cerrado para cada controlador.....	84
Figura 3.8 Respuestas en frecuencia de las diferentes sensibilidades que tiene el sistema en lazo cerrado nominal para ambos controladores.....	85
Figura 3.9 Respuesta en frecuencia de lazo abierto del sistema para cada controlador con el filtro F1	87
Figura 3.10 Respuesta en el tiempo del sistema en lazo cerrado ante una entrada escalón unitario para cada controlador con el filtro F1.....	87
Figura 3.11 Respuesta en frecuencia del sistema en lazo cerrado para cada controlador con el filtro F1	88
Figura 3.12 Respuesta en frecuencia del sistema en lazo cerrado completo para ambos controladores en los tres tipos de sensibilidad.....	89
Figura 3.13 Comandos del radiocontrol para activar y desactivar los motores .	90
Figura 3.14 Esquema general del sistema aéreo no tripulado.....	91
Figura 3.15 Diagrama esquemático	92
Figura 3.16 Cuadri-rotor.....	93
Figura 3.17 Interfaz gráfica de la estación terrestre para computadora y para celular	93
Figura 3.18 Código general del Piccolo para el cuadri-rotor.....	94
Figura 3.19 Diagrama del Stateflow para activar las interrupciones	95
Figura 3.20 Código para activar las interrupciones.....	96
Figura 3.21 Configuración del Memory Copy para definir el registro a leer	96
Figura 3.22 Código para escribir o leer según la interrupción que se activa.....	97
Figura 3.23 Configuración para leer por i2c.....	98
Figura 3.24 Código de unión de las variables que recibe el Piccolo del Crius ..	99
Figura 3.25 Código para estimar la posición angular de alabeo	100
Figura 3.26 Código para calcular y eliminar el offset	100

Figura 3.27 Código para escalar la señal de la guiñada de -180 a 180	102
Figura 3.28 Código para seleccionar modo de vuelo de los canales 1 y 2	102
Figura 3.29 Código para escalar la referencia de los canales 1 y 2.....	103
Figura 3.30 Código para escalar y mantener la referencia del canal 4.....	103
Figura 3.31 Código de la implementación de los controladores.....	104
Figura 3.32 Código para generar los valores necesarios que el Crius necesita para construir el PWM	105
Figura 3.33 Código para transmitir por i2c las ocho mitades de las señales para PWM.....	106
Figura 3.34 Cambios en el código para definir los sensores disponibles en el Crius	108
Figura 3.35 Código comentado de la sección de los controladores.....	109
Figura 3.36 Código añadido para definir las variables a transmitir por i2c.....	109
Figura 3.37 Código agregado para transmitir y recibir las variables según sea el caso	110
Figura 3.38 Código de las subrutinas para tomar lectura de la información enviada por el Piccolo	110
Figura 3.39 Respuesta experimental modificando sólo la referencia de alabeo	111
Figura 3.40 Respuesta experimental modificando la referencia de cabeceo..	112
Figura 3.41 Respuesta experimental modificando la referencia de la guiñada	113
Figura 3.42 Resultados experimentales del cuadri-rotor en vuelo libre	115
Figura 4.1 Diagrama de flujo del proceso de diseño/rediseño de controladores	117
Figura 6.1 Paquetes a instalar del CCS	129
Figura 6.2 Ventana para seleccionar los compiladores en el CCS	130
Figura 6.3 Opciones de la barra de herramientas View	130
Figura 6.4 Ventana donde se define el target necesario para conectar el Piccolo al CCS	131
Figura 6.5 Ventanas para instalar las librerías para los microcontroladores C2000	132

Figura 6.6 Ventana para configuración del target en Matlab	133
Figura 6.7 Ventana para definir las direcciones de las carpetas del CCS, Code Generator y DSP/BIOS	133
Figura 6.8 Ventana donde se selecciona el compilador.....	134
Figura 6.9 Ventana de configuración del Linker.....	135
Figura 6.10 Ventana de configuración del Archiver	135
Figura 6.11 Ventana donde se configura el target que usa el CCS para conectarse al Piccolo	136
Figura 6.12 Selección del compilador Visual C++ 2010.....	139
Figura 6.13 Buscador de librerías del Simulink.....	140
Figura 6.14 Configuración de la unidad de procesamiento a utilizar.....	141
Figura 6.15 Bloques que se pueden utilizar para programar el Piccolo	142
Figura 6.16 Bloque y configuración del GPIO34 como salida digital.....	143
Figura 6.17 Configuración del bloque Pulse Generator	144
Figura 6.18 Ventanas del código terminado, y para configurar dentro del Model Configuration Parameters	145
Figura 6.19 Opción para compilar el código	146
Figura 6.20 Opción para generar un nuevo proyecto en CCS	147
Figura 6.21 Opción de Debug	147
Figura 6.22 Opción para conectar el Piccolo al CSS	148
Figura 6.23 Opción para cargar un programa en el Piccolo.....	148
Figura 6.24 Opción para comenzar a ejecutar el programa que se carga en el Piccolo	148
Figura 6.25 Botones de acceso rápido más usados	149
Figura 6.26 Ventana para seleccionar el área de trabajo	150
Figura 6.27 Ventana que muestra la información de los dispositivos conectados	151
Figura 6.28 Ventana de configuración avanzada donde se escribe el número serial de cada dispositivo	151
Figura 6.29 Localización del GUI Composer dentro del CCS	152

Figura 6.30 Grafica añadida y configuraciones para ver la referencia y la posición estimada	153
Figura 6.31 Bloque de texto con sus respectivas configuraciones.....	154
Figura 6.32 Configuraciones para añadir bloque que modifica la ganancia....	155
Figura 6.33 Configuraciones de la casilla correspondiente al selector de la fuente de la referencia	156
Figura 6.34 Configuración para casillas selectoras del tipo de referencia interna	157
Figura 6.35 Ventana para exportar el proyecto.....	157
Figura 6.36 Ventana de configuración de la aplicación creada.....	158
Figura 6.37 Aplicación final en funcionamiento.....	158

Índice de tablas

Tabla 1-1 Comparación de CDV comerciales	17
Tabla 2-1 Descripción de puertos de conexión.....	31
Tabla 2-2 Especificaciones de control	35
Tabla 2-3 Descripción de los estados del Stateflow.....	51
Tabla 2-4 Media cuadrática (RMS) del error de seguimiento.....	73
Tabla 3-1 Ejes utilizados para la estimación de las posiciones angulares.....	99
Tabla 3-2 Descripción general del código MultiWii 2.2	107

6 Anexos

6.1 Configuración de Matlab y CCS

La siguiente configuración se realizó con el siguiente software: Matlab 2012b, Code Composer Studio (CCS) v5.4, Visual Studio 2010 y controlSuite.

El CCS y el controlSuite se descargan de la página de Texas Instruments (www.ti.com). Al instalar el CCS hay que seleccionar los paquetes de instalación que se requieren, en la ventana Processor Support solo se selecciona la opción para los C2000 que es *C28x 32-bit Real-time*, la opción está encerrada en un rectángulo color rojo en la siguiente figura.

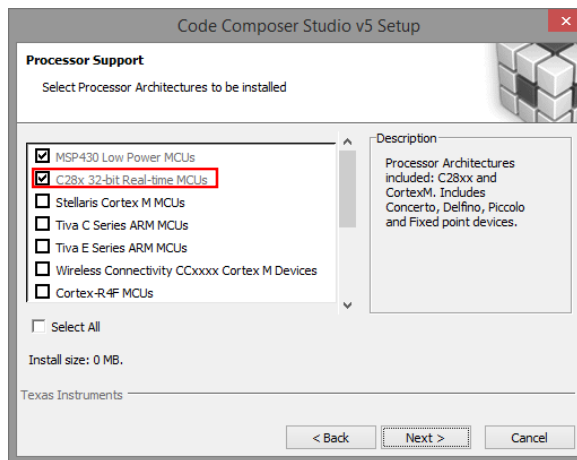


Figura 6.1 Paquetes a instalar del CCS

En la siguiente ventana que se llama Select Components se tiene que seleccionar todas las opciones disponibles, ya que son compiladores de software.

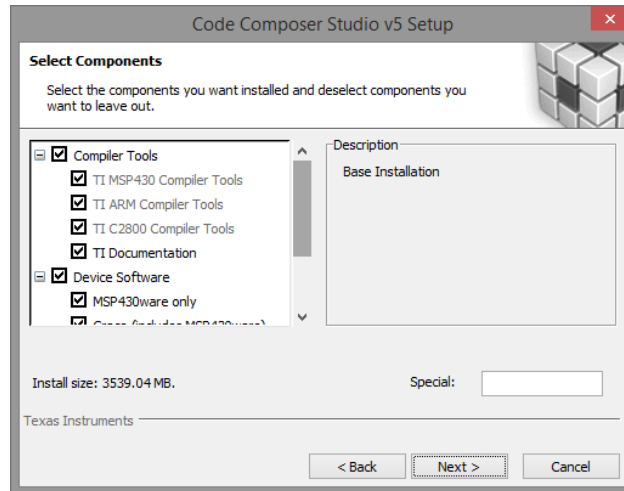


Figura 6.2 Ventana para seleccionar los compiladores en el CCS

Al terminar la instalación se tiene que generar un target para poder conectar la tarjeta al CCS siguiendo los siguientes pasos:

- a) dar clic en View y seleccionar la opción Target Configurations

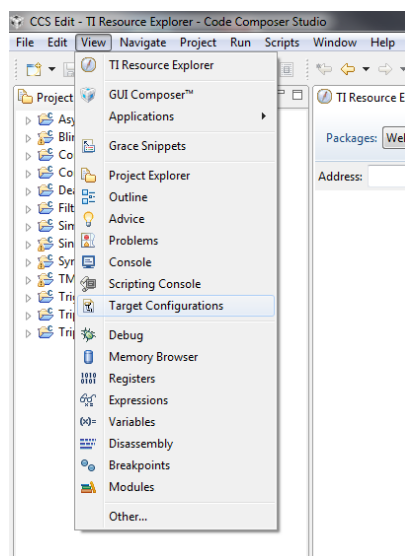


Figura 6.3 Opciones de la barra de herramientas View

- b) se abrirá una pestaña en la cual se tiene que seleccionar en la parte de *Connection* la opción *Texas Instruments XDS100v1 USB Emulator*, y seleccionar el tipo de tarjeta que se esté utilizando, en este caso se selecciona *controlSTICK – Piccolo F28069* y se le da clic en el botón *save*.

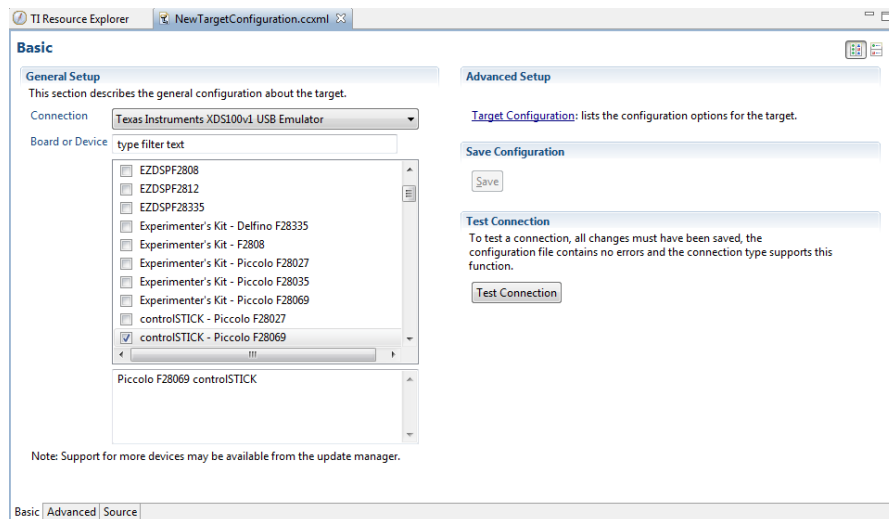


Figura 6.4 Ventana donde se define el target necesario para conectar el Piccolo al CCS

Después se instala el controlSUITE y el Visual Studio 2010.

6.1.1 Configuración de Matlab

En la programación del código en Simulink-Matlab se necesita el toolbox Embedded Coder que incluye los bloques necesarios para la familia C2000 de Texas Instruments. Para las versiones más recientes de Matlab se tienen que

instalar utilizando el comando *targetinstaller*, en la ventana que se abre se selecciona la opción de instalar desde internet.

Después seleccionar e instalar la opción Texas Instruments C2000.

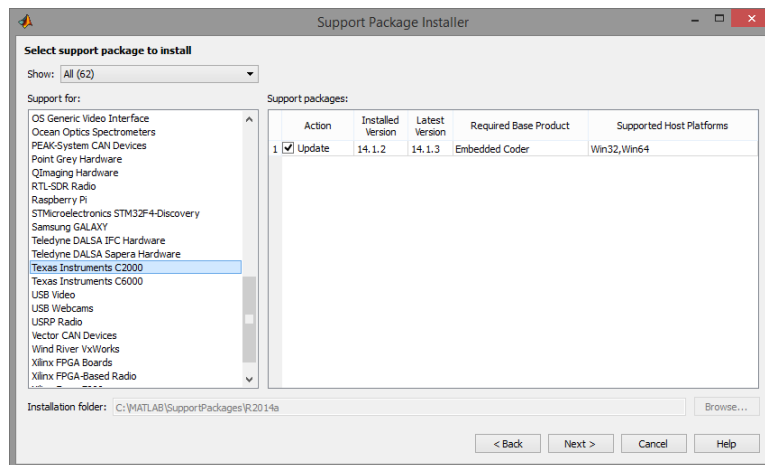
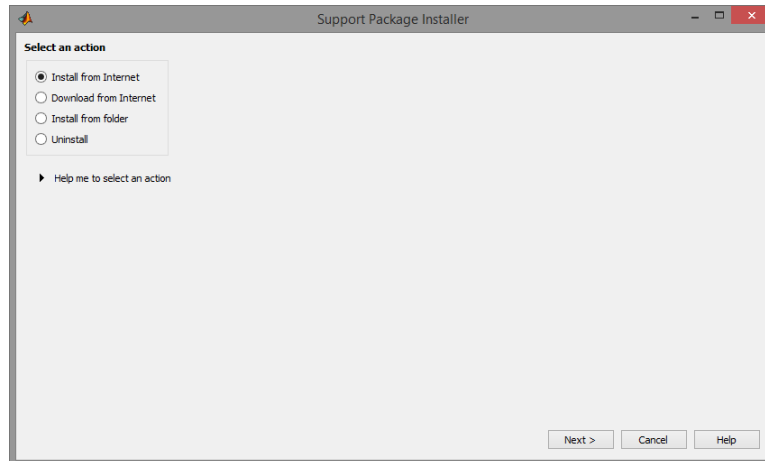


Figura 6.5 Ventanas para instalar las librerías para los microcontroladores C2000

El siguiente paso es para generar el “target” en Matlab con el comando *xmakefilesetup*

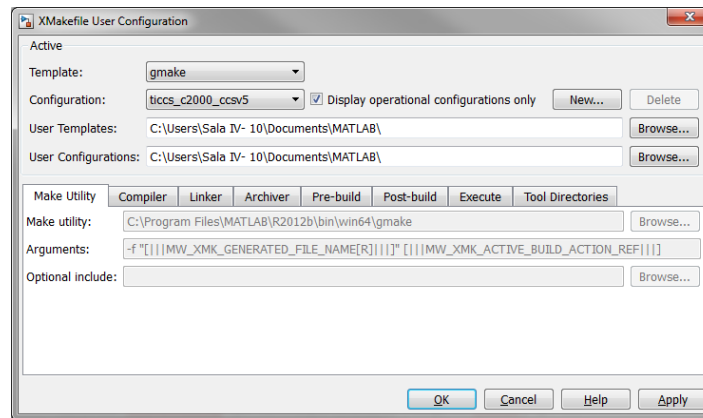


Figura 6.6 Ventana para configuración del target en Matlab

Dentro de esta ventana hay que desmarcar la casilla “Display operational configurations only” y seleccionar en la opción “Configuration”, que está a la izquierda de la casilla desmarcada, la opción “ticcs_C2000_ccsv5”. Después se selecciona la pestaña “Tool Directories” y agregar las siguientes direcciones en los paths como se muestra en la siguiente imagen.

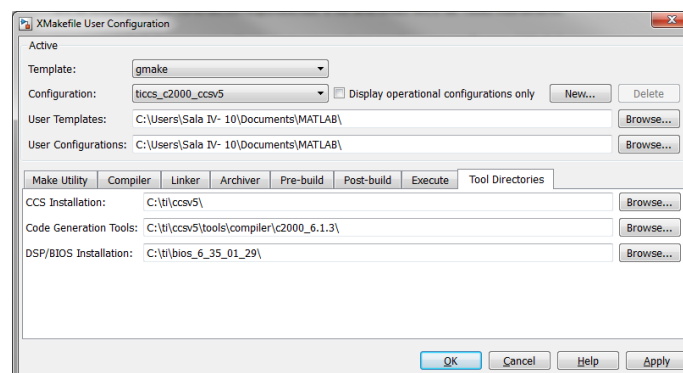


Figura 6.7 Ventana para definir las direcciones de las carpetas del CCS, Code Generator y DSP/BIOS

Los paths o direcciones de las carpetas donde se encuentran instalados el CCSv5, el compilador y el DSP/BIOS, tiene que ser como los que aparecen en la Figura 6.7.

Ahora se pulsa sobre el botón Apply y luego en New... en la parte superior para que nos muestre una ventana en donde se puede cambiar el nombre o dejarlo como está, ya que genera una copia con las configuraciones hechas en los pasos anteriores y además se podrán modificar las demás pestañas.

Ya teniendo el nuevo archivo en el cual se pueden modificar las pestañas se tiene que verificar que tengan lo siguiente:

En la pestaña Compiler, en la sección de compiler se tiene que tener algo como:

C:\TI\ccsv5\tools\compiler\c2000\bin\cl2000

Y en la sección de arguments dentro de la misma pestaña se copia lo siguiente:

-I"C:\TI\ccsv5\tools\compiler\C2000\include" -fr"[MW_XMK_DERIVED_PATH_REF]"

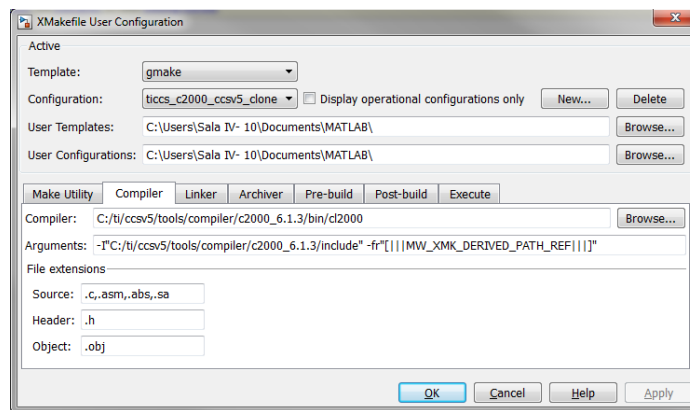


Figura 6.8 Ventana donde se selecciona el compilador

En la pestaña Linker se agrega la siguiente dirección:

C:\TI\ccsv5\tools\compiler\C2000\bin\cl2000

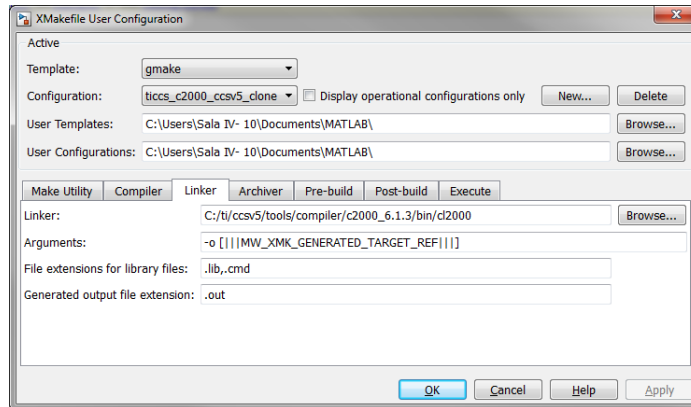


Figura 6.9 Ventana de configuración del Linker

Y en la pestaña Archiver se le agrega la siguiente dirección:

C:\TI\ccsv5\tools\compiler\C2000\bin\ar2000

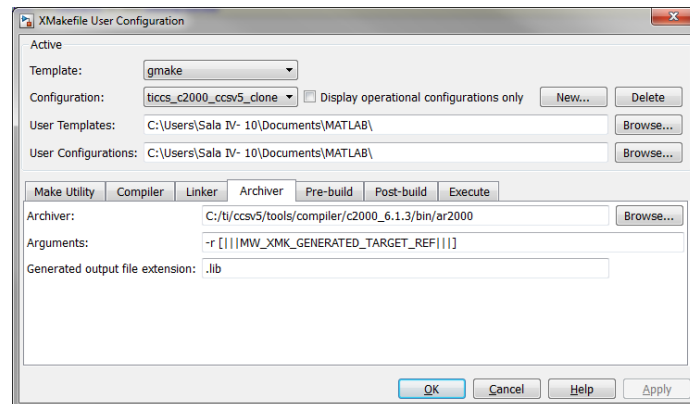


Figura 6.10 Ventana de configuración del Archiver

En la pestaña Execute se tiene que modificar en las dos secciones, en la primera sección llamada Execute tool se tiene que remplazar “echo” por la siguiente dirección:

```
C:\ti\ccsv5\ccs_base\scripting\bin\dss.bat
```

Y en la sección arguments son tres direcciones cada una entre comillas y separadas por un espacio. La primera dirección es en donde se encuentra el archivo runProgram.js, el segundo es en donde se encuentra el target generado por el CCS y el tercer argumento tiene que ser el siguiente:
"[[[[MW_XMK_GENERATED_TARGET_REF[E]]]]]"

Por ejemplo:

```
"C:\Program Files\MATLAB\R2012b\toolbox\idelink\extensions\ticcs\ccsdemos\runProgram.js"
```

```
"C:\Users\Sala IV- 10\ti\CCSTargetConfigurations\NewTargetConfiguration.ccxml"
```

```
"[[[[MW_XMK_GENERATED_TARGET_REF[E]]]]]"
```

Por último se le da clic en OK y con esto se termina de generar el target en Matlab.

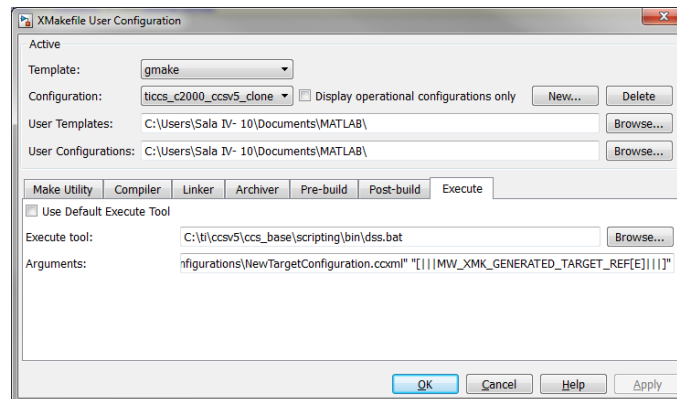


Figura 6.11 Ventana donde se configura el target que usa el CCS para conectarse al Piccolo

El siguiente paso es ingresar en Matlab el comando *checkEnvSetup('ccsv5','F28069','setup')* y al hacerlo se abre una ventana para seleccionar las carpetas de las siguientes direcciones en el siguiente orden:

C:\ti\ccsv5

C:\ti\ccsv5\tools\compiler\c2000_6.1.3

C:\ti\bios_6_35_01_29

C:\ti\controlSUITE\device_support\f2806x\v120

C:\ti\controlSUITE\libs\utilities\flash_api\2806x\v100

Hay que aclarar que tanto lo que se hizo con el comando *xmakefilesetup*, como con el comando *checkEnvSetup('ccsv5','F28069','setup')* se hacen solamente una vez ya que es la parte de la configuración del Matlab.

Para comprobar que está guardada la configuración hecha en el paso anterior se escribe el siguiente comando *checkEnvSetup('ccsv5','F28069','check')* y se nota

que al final de cada sección se encuentran las carpetas agregadas tal como se muestra a continuación:

```
>> checkEnvSetup('ccsv5','F28069','check')
```

1. CCSv5 (Code Composer Studio)

Your version : 5.4.0

Required version: 5.0 or later

Required for : Code Generation

TI_DIR="C:\ti\ccsv5"

2. CGT (Texas Instruments C2000 Code Generation Tools)

Your version : 6.1.3

Required version: 5.2.1 to 6.0.2

Required for : Code generation

C2000_CGT_INSTALLDIR="C:\ti\ccsv5\tools\compiler\c2000_6.1.3"

3. DSP/BIOS (Real Time Operating System)

Your version : 6.35.01.29

Required version: 5.33.05 to 5.41.11.38

Required for : Code generation

CCSV5_DSPBIOS_INSTALLDIR="C:\ti\bios_6_35_01_29"

4. XDC Tools (eXpress DSP Components)

Your version :

Required version: 3.16.02.32 or later

Required for : Code generation

5. CCS (Code Composer Studio)

Your version :

Required version: 4

Required for : Automation and Code Generation

6. 2806x C/C++ Header Files

Your version : 1.20

Required version: 1.20

Required for : Code generation

DSP2806x_INSTALLDIR="C:\ti\controlSUITE\device_support\2806x\v120"

7. Flash Tools (TMS320F2806x Piccolo(TM) Flash API)

Your version : 1.00

Required version: 1.00

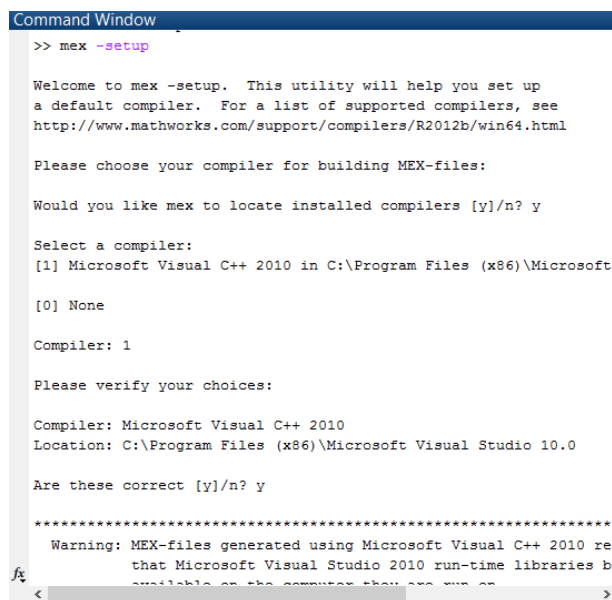
Required for : Flash Programming

FLASH_2806X_API_INSTALLDIR="C:\ti\controlSUITE\libs\utilities\flash_api\2806x\v100"

>>

NOTA: se puede observar que no todas las secciones cuentan con una carpeta, esto es porque no aplican para la versión 5 del Code Composer.

Para terminar la configuración en Matlab es necesario agregar el compilador que es el Visual Studio, para hacer esto se introduce la siguiente instrucción `mex -setup` y se selecciona el compilador Microsoft Visual C++ 2010.



```
Command Window
>> mex -setup

Welcome to mex -setup. This utility will help you set up
a default compiler. For a list of supported compilers, see
http://www.mathworks.com/support/compilers/R2012b/win64.html

Please choose your compiler for building MEX-files:

Would you like mex to locate installed compilers [y]/n? y

Select a compiler:
[1] Microsoft Visual C++ 2010 in C:\Program Files (x86)\Microsoft
[0] None

Compiler: 1

Please verify your choices:

Compiler: Microsoft Visual C++ 2010
Location: C:\Program Files (x86)\Microsoft Visual Studio 10.0

Are these correct [y]/n? y

*****
Warning: MEX-files generated using Microsoft Visual C++ 2010 require
that Microsoft Visual Studio 2010 run-time libraries be
available on the computer they are run on
*****
```

Figura 6.12 Selección del compilador Visual C++ 2010

En caso de tener problemas con el comando *mex* una solución es limpiar el área de trabajo y la ventana de comandos con *clear* y *clc* respectivamente. En caso de no solucionarse el problema la otra opción es reiniciar Matlab.

6.1.2 Ejemplo de programación

Con la configuración terminada ya se pueden generar códigos en Simulink/Matlab para programar el Piccolo de la siguiente forma.

NOTA: este método esta realizado para la versión de Matlab 2012b, para otra versión más reciente del Matlab se tiene que investigar como configurar el *Target Preferences*.

Para empezar hay que abrir un nuevo modelo de Simulink y buscar en las librerías la que se llama *Embedded Coder* después se selecciona *Embedded Targets* y se agrega el bloque *Target Preferences*.

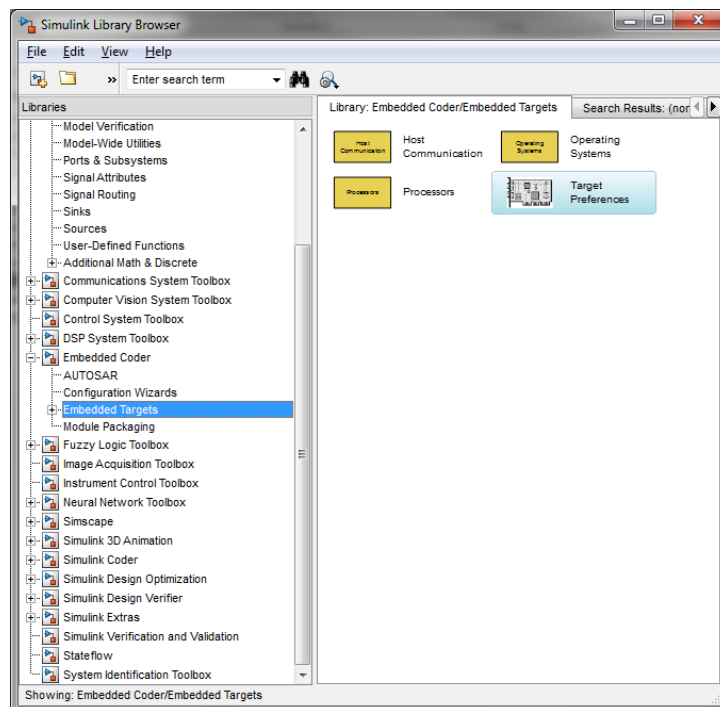


Figura 6.13 Buscador de librerías del Simulink

Al agregar el bloque al área de trabajo del modelo de Simulink se abre una ventana como lo muestra la Figura 6.14, en la cual se tiene que seleccionar la versión utilizada del Code Composer Studio en este caso la versión 5 y en el tipo de tarjeta que se va a utilizar se selecciona la *TI F28069 (boot from flash)* para que la programación sea en la memoria flash y de esta manera se almacene el código en el Piccolo. La otra opción es *TI F28069* pero el código se programa en la memoria RAM, esto trae como consecuencia que al desconectarlo de la alimentación se pierda la programación y cuando se vuelve a conectar se ejecute el código almacenado en la memoria flash.

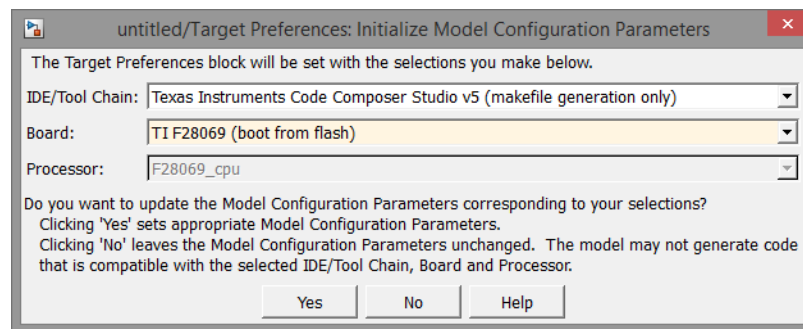


Figura 6.14 Configuración de la unidad de procesamiento a utilizar

Dentro de la librería *Embedded Targets* se encuentra *Processors* que es en donde están los diferentes dispositivos que se pueden programar, entre ellos se

encuentran los de la familia C2000 de Texas Instruments. En este caso se selecciona el C2806X porque es el modelo que se está utilizando.

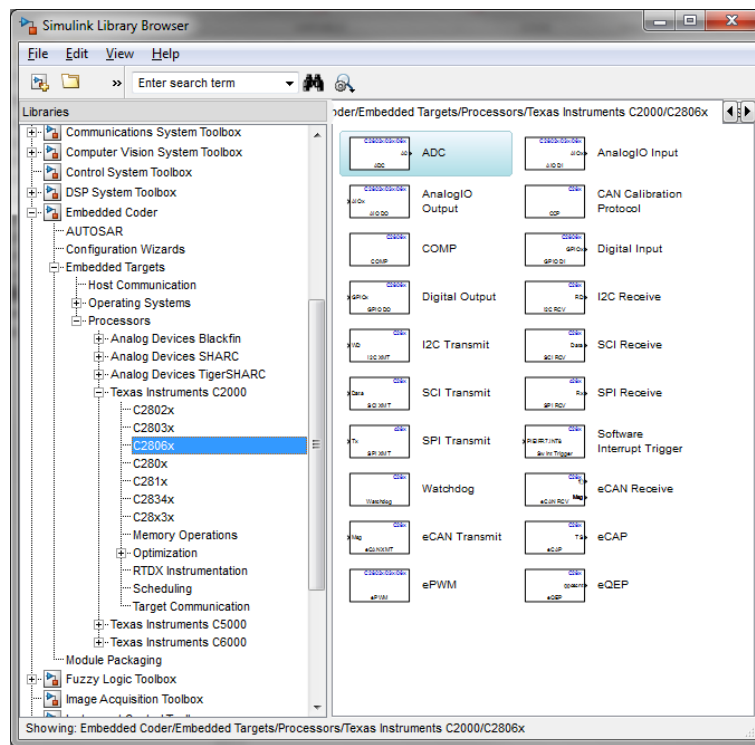


Figura 6.15 Bloques que se pueden utilizar para programar el Piccolo

Para explicar el proceso de programación, se va a comenzar con un ejemplo sencillo el cual consta de hacer encender y apagar un LED a un determinado tiempo de muestreo.

Primero hay que agregar el bloque *Digital Output* y abrir las opciones haciendo doble clic sobre el bloque desmarcar la casilla seleccionada y seleccionar el GPIO34 que es el LED disponible en la tarjeta.

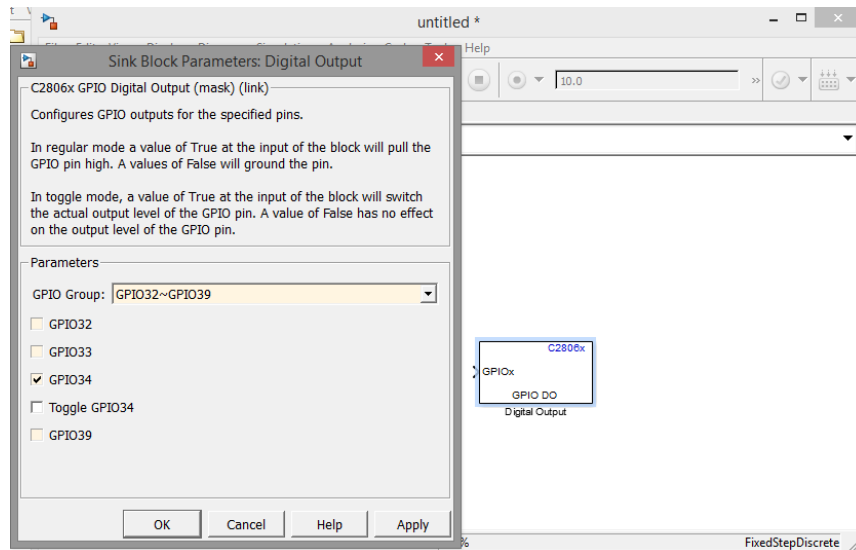


Figura 6.16 Bloque y configuración del GPIO34 como salida digital

Después se agrega el bloque *Pulse Generator* y se conecta al bloque anterior. La configuración que se tiene que hacer es modificar el *Pulse Width* a 50% para que la señal sea simétrica según en periodo determinado, el periodo de la señal

se modifica en la opción *Period* en este caso se define 1 segundo. La configuración queda de la siguiente manera.

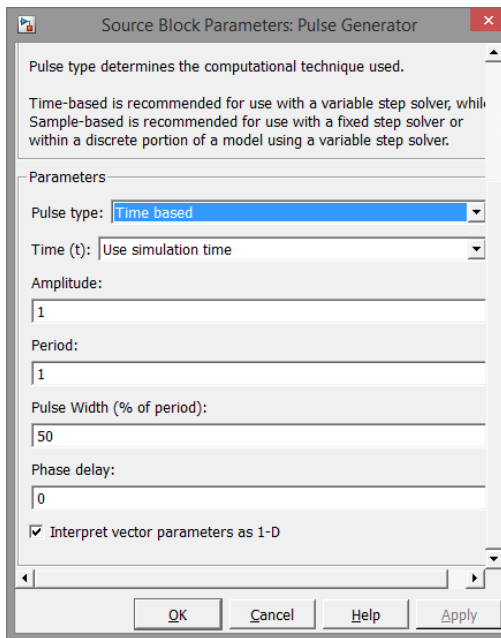


Figura 6.17 Configuración del bloque Pulse Generator

El siguiente paso es abrir la opción *Model Configuration Parameters* que está en *Simulation* o con el icono en forma de engrane de color gris. Dentro de esta opción se mostrará una ventana que tiene muchas opciones de configuración. Dentro de esta ventana en la opción *Code Generation* se tiene que escoger el archivo *idelink_ert.tlc* en el apartado *Target Selection* y hacer clic en *Apply* para guardar este cambio. Ahora en IDE Link hay que seleccionar la opción *Build* para

que *Simulink* solo compile el código del programa una vez hecho esto se guardan los cambios y se cierra la ventana.

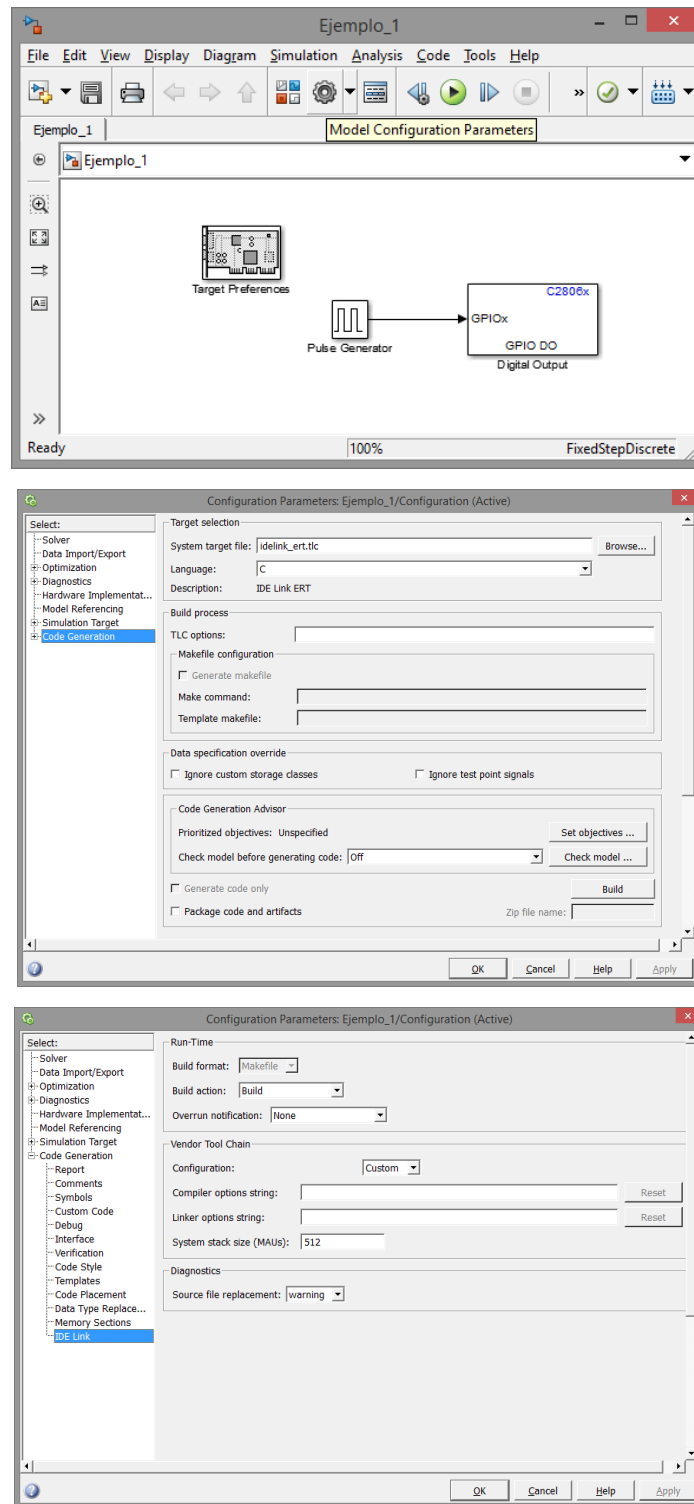


Figura 6.18 Ventanas del código terminado, y para configurar dentro del Model Configuration Parameters

Por último se hace clic en la opción *Build Model* que se encuentra en la barra de herramientas *Code* en la opción *C/C++ Code* o en el icono que tiene la forma de un rectángulo con puntos en el interior con tres flechas apuntando hacia abajo de color azul. Y con esto se termina la parte de programación en Simulink/Matlab.

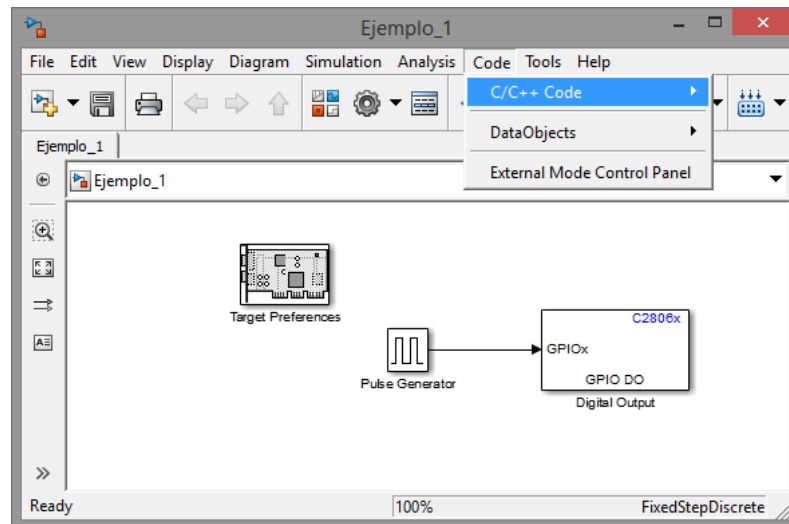


Figura 6.19 Opción para compilar el código

La parte final de la programación se realiza en el Code Composer Studio que consiste en cargar el código generado en el Piccolo. Para empezar hay que agregar el código como un proyecto existente porque tiene la ventaja de que en caso de hacer alguna modificación en el código cuando se vuelva a compilar, con el *Piccolo* conectado en el CCS y pausado en la ejecución del código, se programe de manera automática inmediatamente después de terminar la compilación del programa en Simulink. Como primer paso es generar un nuevo proyecto del tipo *C/C++* con un código existente, después en la ventana de importación buscar la carpeta que contiene el archivo generado por Simulink que

se almacena en la carpeta de Matlab y tiene como característica que al final del nombre `_ticcs` de esta manera se genera el proyecto en CCS.

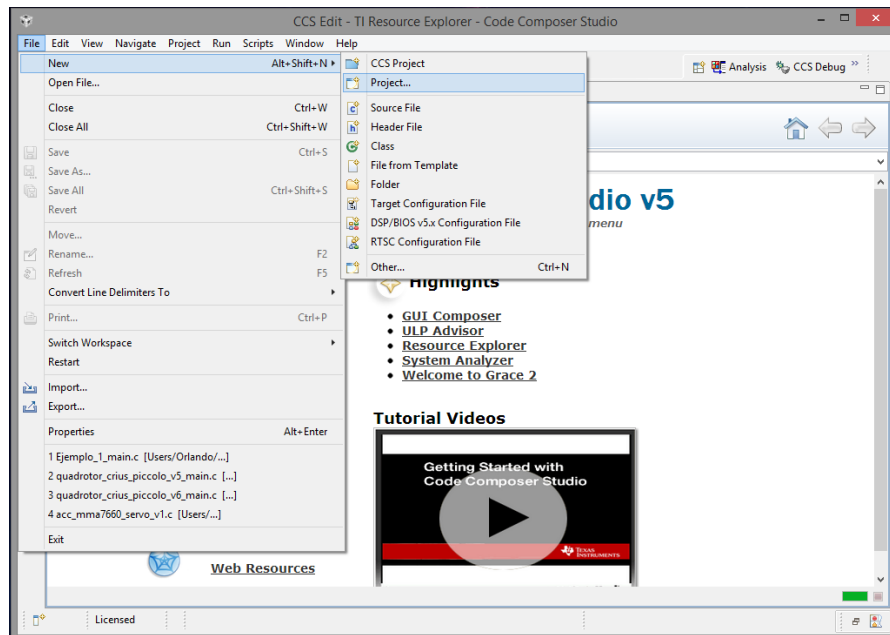


Figura 6.20 Opción para generar un nuevo proyecto en CCS

El siguiente paso es cargar el código por lo cual se tiene que pasar al modo de *Debug*, esta opción se encuentra en el apartado *Run* de la barra de herramientas.

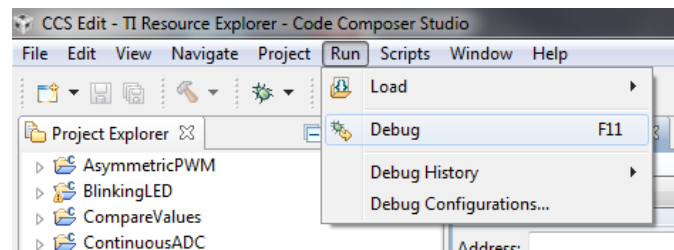


Figura 6.21 Opción de Debug

El Piccolo conectado a la computadora necesita realizar la comunicación con el CCS y para que se logre esto es necesario definir el target, hecho en el apartado de configuración inicial, como predeterminado. La forma en que se hace es simplemente dar clic derecho sobre el target configurado y activar la opción *Set*

as *Default*. Ahora lo único que falta es activar la opción *Connect Target* que se encuentra dentro del apartado *Run* de la barra de herramientas.

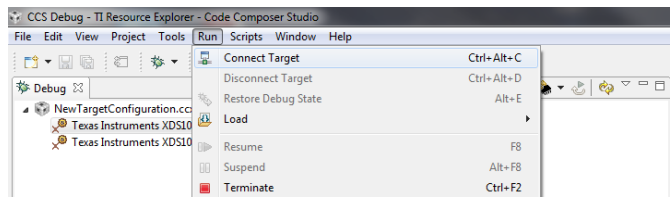


Figura 6.22 Opción para conectar el Piccolo al CSS

Para programar el código deseado es por medio de *Load Program* que también se encuentra en el apartado *Run*. Buscar en la carpeta que genera Simulink del código el archivo que lleva el nombre del código con la extensión *.out* y dar clic en *Ok*, solo hay que esperar a que termine de programar el dispositivo.

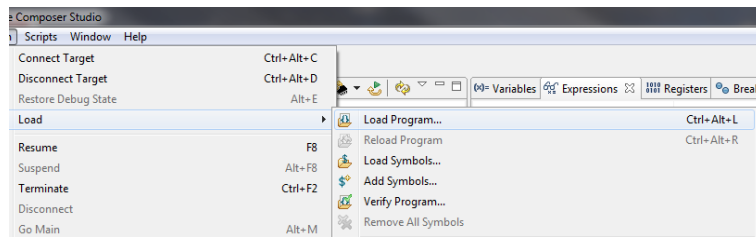


Figura 6.23 Opción para cargar un programa en el Piccolo

Una vez que se haya programado solamente se da clic en *Resume* para que el Piccolo comience a ejecutar el código.

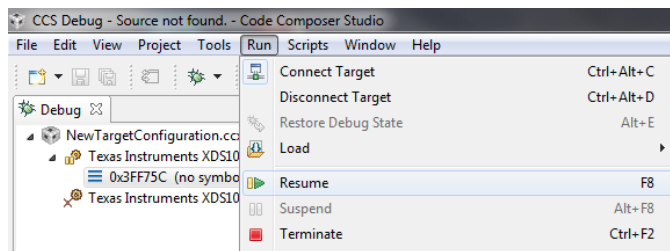


Figura 6.24 Opción para comenzar a ejecutar el programa que se carga en el Piccolo

También están los botones para realizar la programación de una manera más rápida los cuales se muestran a continuación:

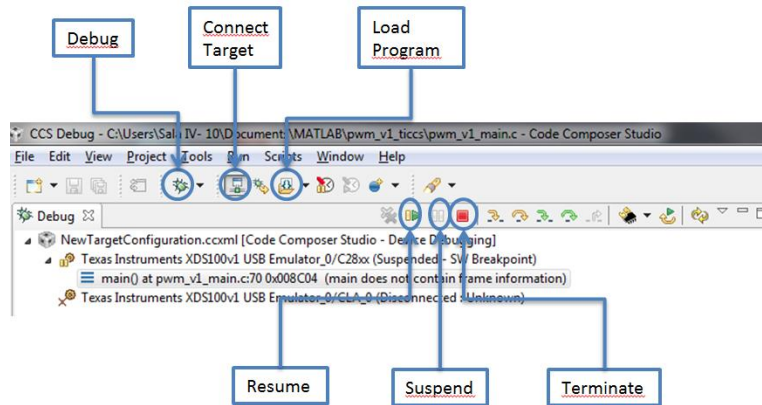


Figura 6.25 Botones de acceso rápido más usados

Hay que aclarar la diferencia de *Suspend* y *Terminate*. La primera es para detener o pausar el programa que se esté ejecutando pero manteniendo al *Piccolo* conectado y poder cargar otro programa o reiniciar el mismo. En la segunda opción sirve para detener el programa, desconectar el *Piccolo* del CCS y salir del modo *Debug*.

6.1.3 Simulación simultánea

En algunas ocasiones es necesario conectar dos dispositivos en una misma computadora y trabajar con ellos al mismo tiempo, por ejemplo, cargar algún programa diferente en cada uno y ejecutarlos. El método o los pasos a seguir para poder trabajar de esta forma son los siguientes:

1. Abrir dos ventanas del CCS.
2. Antes de que se abra el CCS aparece la ventana de la Figura 6.26, en la cual tiene que tener diferente *workspace* para cada CCS que se abra, es decir, se tiene que cambiar la dirección de la carpeta que aparece con la

ayuda del botón *Browse...*, este cambio sólo se realiza en una, porque la primera que se abre se puede dejar con la carpeta por defecto.

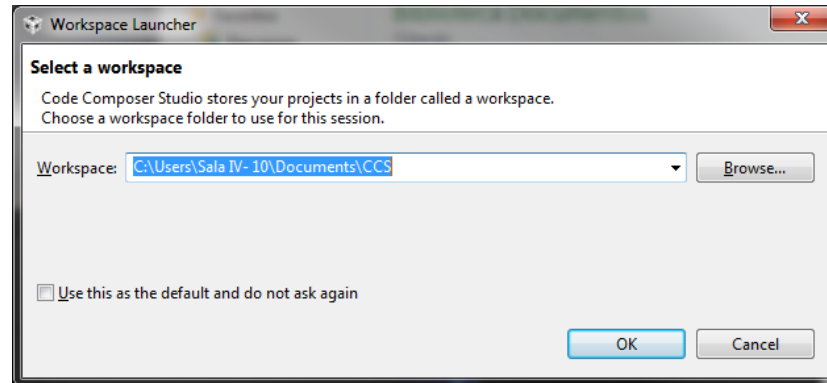


Figura 6.26 Ventana para seleccionar el área de trabajo

3. En cada CCS hay que configurar un target con diferente nombre y con las siguientes dos variaciones:
 - a. En la casilla de *Connection* uno tendrá la opción *Texas Instruments XDS100v1 USB emulator* y el otro *Texas Instruments XDS100v2 USB emulator*.
 - b. Cada target contará con un número serial, que es único para cada Piccolo.
4. Para obtener el número serial hay que seguir los siguientes pasos:
 - a. Conectar ambos Piccolos a la computadora.
 - b. Para obtener el número serial se tiene que buscar en la dirección *C:/ti/ccsv5/ccs_base/common/uscif* el archivo *wdx100serial.exe* y ejecutarlo. Este ejecutable despliega la información de los dispositivos conectados, donde la tercera columna corresponde al

número serial. En la Figura 6.27 se muestra un ejemplo de como aparece la información.

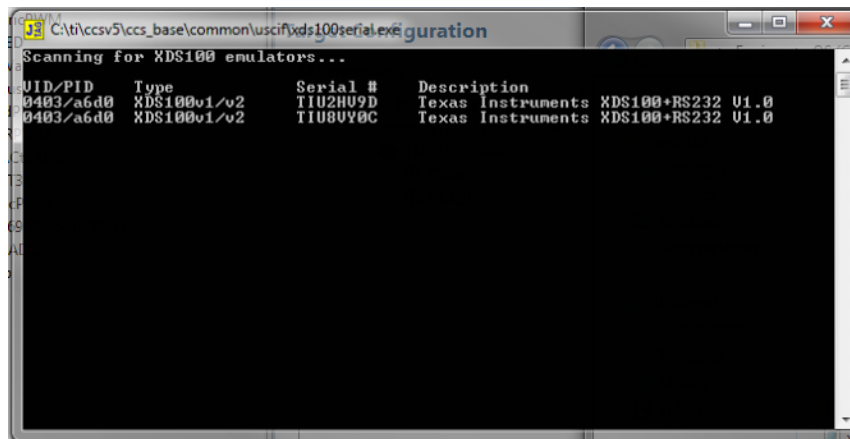


Figura 6.27 Ventana que muestra la información de los dispositivos conectados

- c. Dentro del Target Configuration seleccionar el modelo de la tarjeta a utilizar, después se cambia a la pestaña Advanced. Al pulsar en la opción All Connections van a aparecer las propiedades de la conexión según el tipo de dispositivo seleccionado.
- d. Cambia la opción del *Emulator Selection* a *Select by serial number* para que aparezca la casilla en donde se tiene que ingresar el número serial.

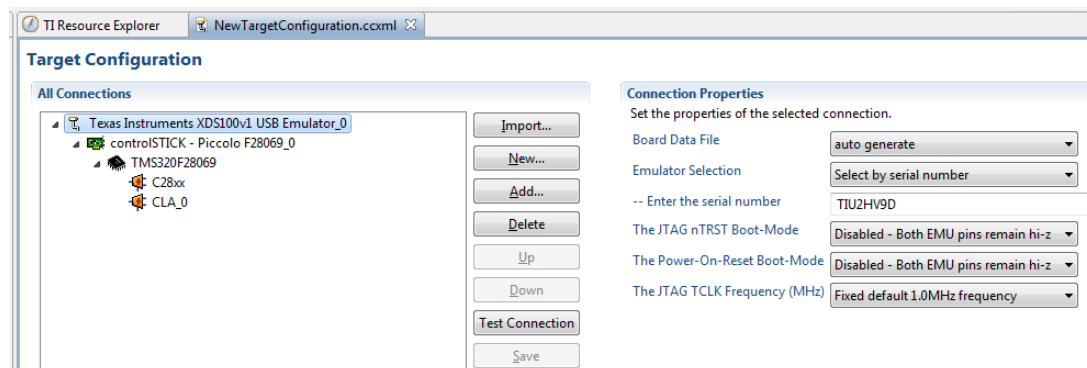


Figura 6.28 Ventana de configuración avanzada donde se escribe el número serial de cada dispositivo

6.2 GUI Composer

Este es un complemento del CCS para crear interfaces y poder interactuar con el programa en tiempo real. Consta de dos partes, la primera ya está incluida en el CCS y es para el diseño de la interfaz gráfica, mientras que la segunda parte llamada GUI Composer Run Time se tiene que descargar e instalar ya que se necesita para poder correr las aplicaciones hechas.

Como ejemplo se va a diseñar una aplicación para el PVTOL en el cual muestre una gráfica con la referencia y la salida, así como los selectores correspondientes que tiene el código.

Después de programar el Piccolo se puede comenzar a generar la interfaz gráfica. El GUI Composer se encuentra en *View* de la barra de herramientas del CCS al seleccionarlo con un clic se abre una ventana.

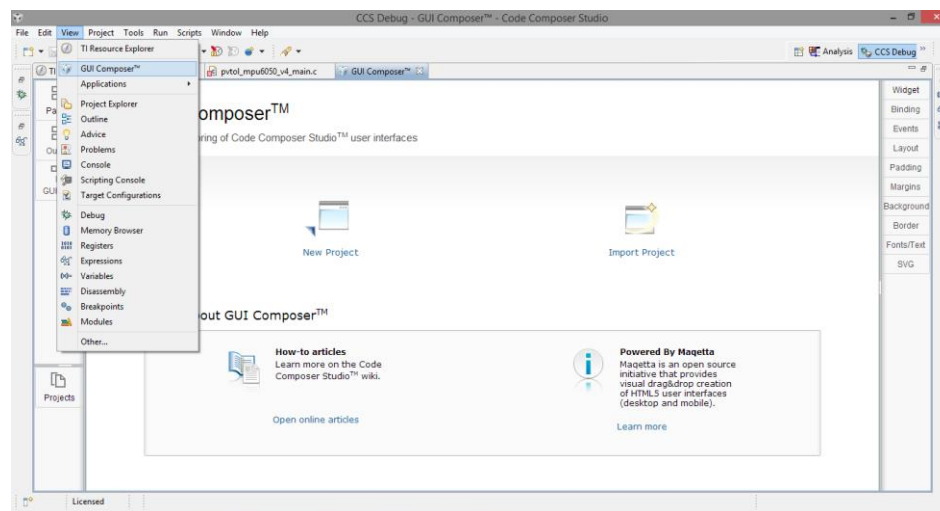
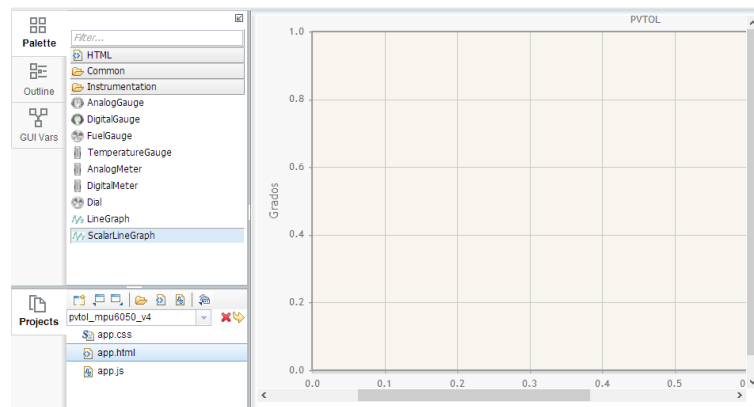


Figura 6.29 Localización del GUI Composer dentro del CCS

Se selecciona la opción de crear nuevo proyecto y se le da un nombre, se recomienda que todos los archivos generados tengan el mismo nombre para evitar confusiones. Al nombrar el proyecto aparece una ventana que es el editor, aquí es en donde se crea la interfaz.

Lo primero que se agrega es la gráfica, esta grafica se encuentra en la carpeta *Instrumentation* que se encuentra a la izquierda, para agregarla se selecciona *ScalarLineGraph* con un clic y en la sección en blanco se dibuja un rectángulo del tamaño que se desee. Después de que se genera la gráfica es necesario configurarla para que muestre las variables de referencia y de salida. Con la gráfica seleccionada se escoge la pestaña *Binding* ubicada en el lado derecho de la pantalla y en las casillas *Value0* y *Value1* se escriben las variables correspondientes a la referencia y a la salida del sistema, es decir, la posición estimada. En la pestaña *Widget* se configuran las propiedades de la gráfica como el rango de los ejes, el título de la gráfica, nombre de los ejes entre otras.



The image shows two screenshots of the widget configuration panels for the 'gc.dijit.ScalarLineGraph #widget_335' widget. The left screenshot shows the 'Widget' tab with various properties: Title: PVTOL, Buffer Length: 200, X-Axis Minimum Value: 0, X-Axis Maximum Value: 10, X-Axis Label: Tiempo, Y-Axis Minimum Value: 0, Y-Axis Maximum Value: 10, Y-Axis Label: Grados, Value 0 Label: Ref, Value 1 Label: Pos, Series 0 Color: blue, Series 1 Color: green. The right screenshot shows the 'Binding' tab with 'Value 0' set to 'lec_rc' and 'Value 1' set to 'posicion'. Other tabs like 'Events', 'Layout', 'Padding', 'Margins', 'Background', 'Border', 'Fonts/Text', and 'SVG' are also visible.

Figura 6.30 Grafica añadida y configuraciones para ver la referencia y la posición estimada

El siguiente elemento que se agrega es un *TextBox* que se encuentra en la carpeta *Common*, este cuadro de texto es muy importante ya que será el que defina el tiempo de muestreo de la aplicación. El valor que se le asigna en la pestaña *Binding* es *value:pm.\$refresh_interval* esta instrucción es para poder modificar el tiempo de muestreo de la aplicación. En la pestaña *Widget* se define el valor que tiene que ser 100 (son milisegundos, 100 es el mínimo valor permitido).

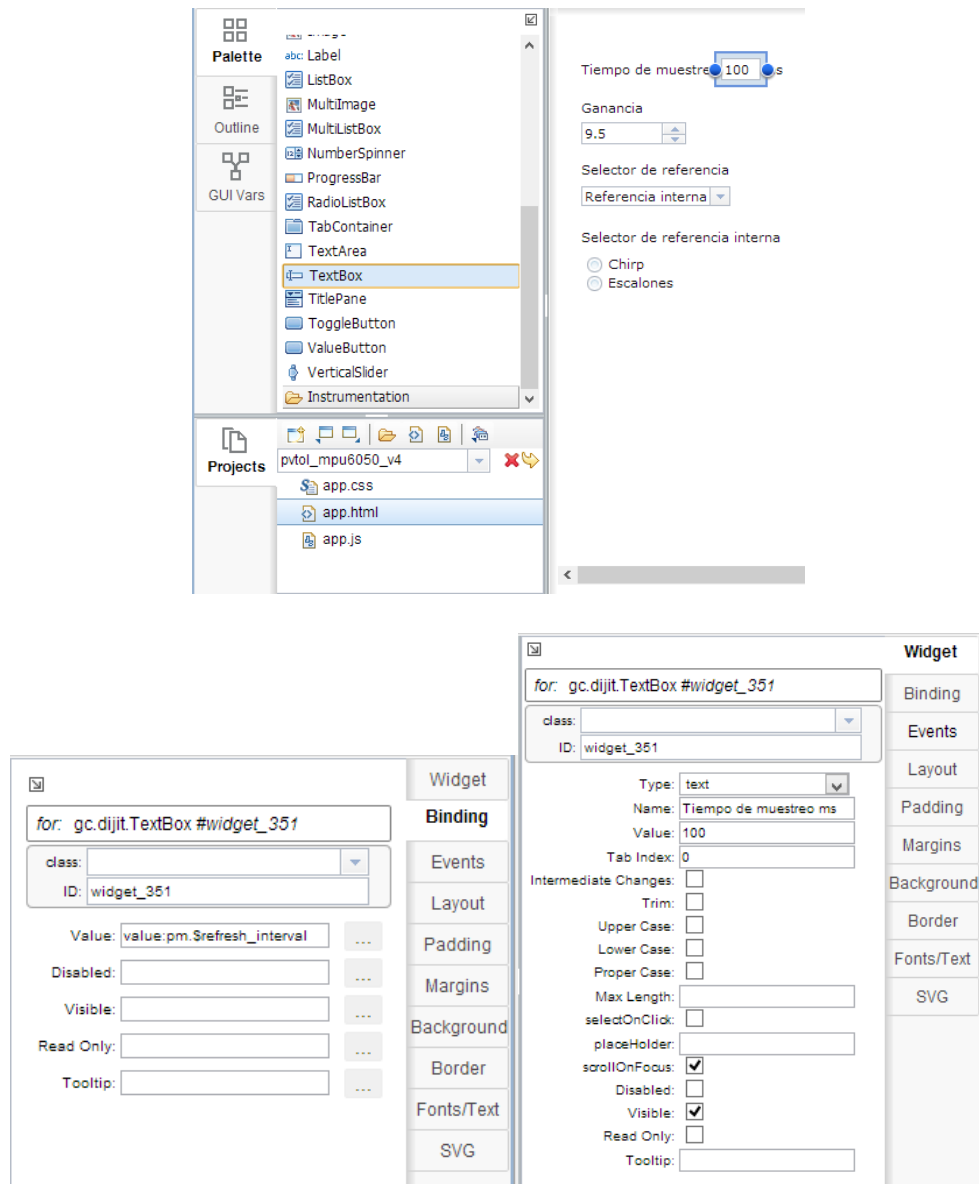


Figura 6.31 Bloque de texto con sus respectivas configuraciones

Para modificar la ganancia se agrega un *NumberSpinner* en el cual se asigna la variable que en este caso es *pvtol_mpu6050_v4_P.prop_gain_Value*, y dentro de *Widget* se define la casilla *Small Delta* como 0.1 para que los cambios sean de 0.1

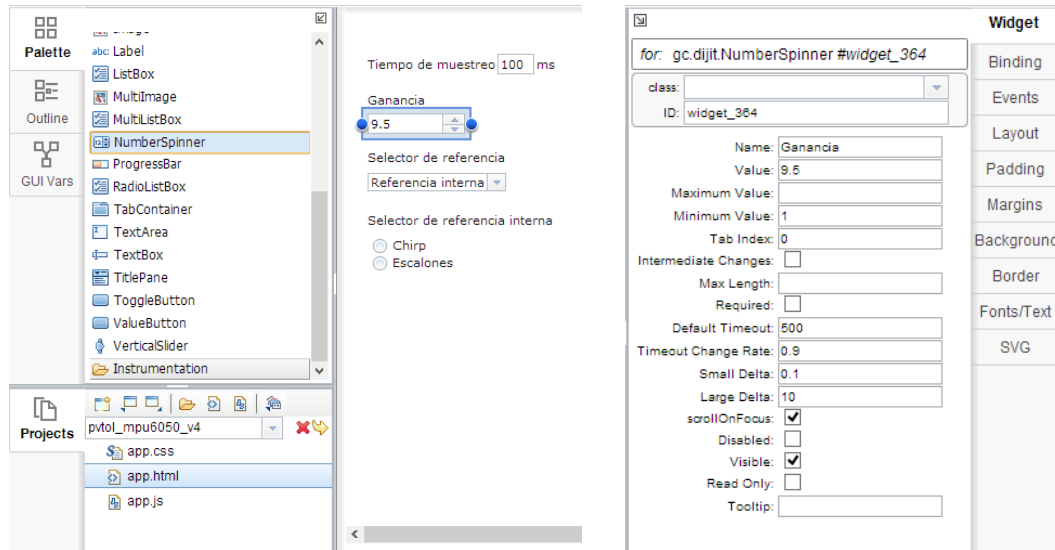


Figura 6.32 Configuraciones para añadir bloque que modifica la ganancia

Para seleccionar la fuente de la referencia si es externa o interna se agrega un *DropDownSelect* y se escriben las opciones que son referencia externa y referencia interna. La configuración que tiene que tener es: en la pestaña *Binding* se agrega la variable en la casilla *Selected Value* en este caso es *pvtol_mpu6050_v4_P.sel_ref_Value*, y dentro de la pestaña *Widget* se puede observar en la parte de abajo una casilla llamada *Labels* que contiene los nombres de las opciones añadidas y la casilla *Values* se escriben los valores que van a tomar las opciones dependiendo de la configuración de los selectores en Simulink, en este caso 1 es para la referencia externa mientras que para la interna es 0, estos números se escriben separados por una coma.

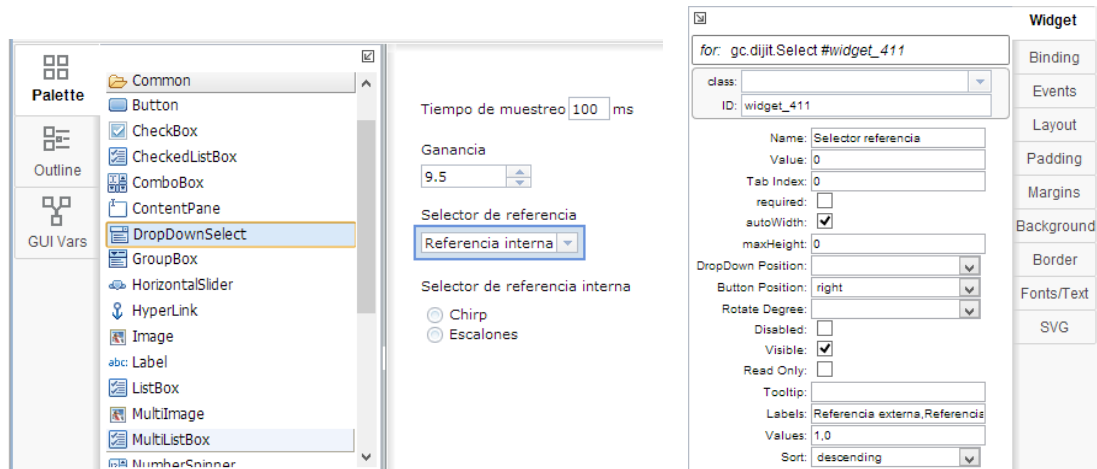


Figura 6.33 Configuraciones de la casilla correspondiente al selector de la fuente de la referencia

Por último se agrega un *RadioListBox* para seleccionar que tipo de referencia interna toma el sistema, ya sea la señal Chirp o la serie de escalones. Cuando se agrega se escriben las opciones para las señales. Las configuraciones que hay que hacer son: en la pestaña *Binding* se agrega la variable correspondiente al bloque que selecciona el tipo de referencia interna, que en este caso es *pvtol_mpu6050_v4_P.tip_ref_Value* en la casilla de *Selected Value* y en la casilla *Read Only* se escribe la variable que selecciona la fuente de la referencia que en este caso es *pvtol_mpu6050_v4_P.sel_ref_Value*, en la pestaña *Widget* al igual que en el elemento anterior las ultimas opciones se encuentran las casillas con el nombre de las opciones agregadas y en la última llamada *Values* se escriben el valor que toman las opciones al ser seleccionadas en la interfaz, 0 para el Chirp y 1 para los escalones.

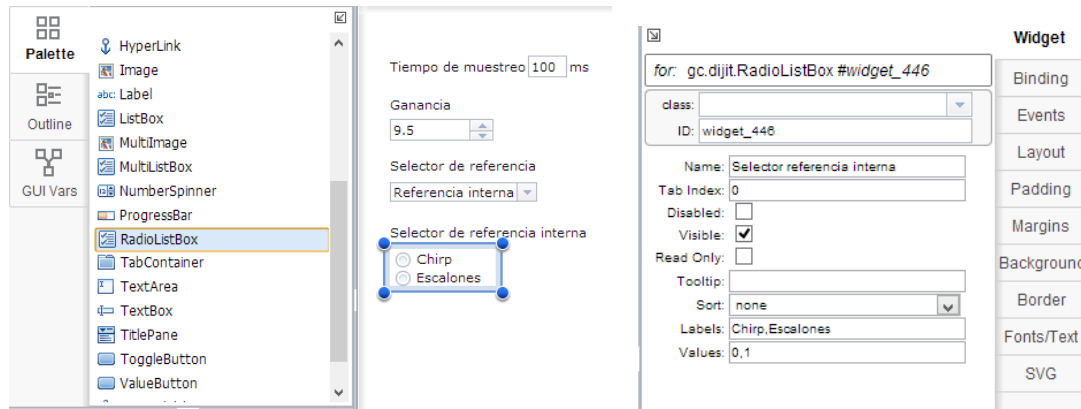


Figura 6.34 Configuración para casillas selectoras del tipo de referencia interna

Para guardar el proyecto se da clic en *Export project* y aparece una ventana donde en la primera opción se escoge la dirección en donde guardar el proyecto, en la segunda opción el tipo de conexión que tiene el Piccolo con el CCS, en la tercera el tipo de microcontrolador que se está utilizando y en la última opción se selecciona el archivo a programar que es generado por Simulink con extensión .out, la configuración la muestra la siguiente imagen:

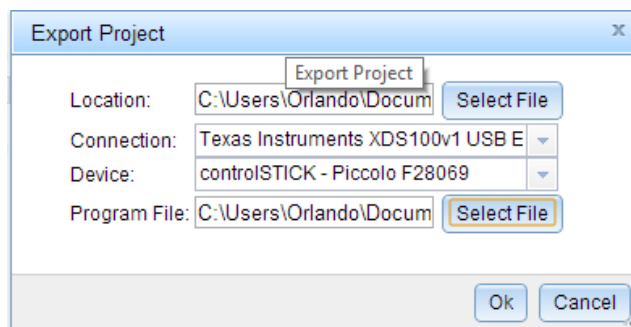


Figura 6.35 Ventana para exportar el proyecto

El siguiente paso es buscar el proyecto en la dirección donde se guardó, el proyecto es un archivo comprimido .zip, este se descomprime en una carpeta con el nombre del proyecto y se copia en la siguiente dirección:

C:\ti\guicomposer\webapps

Lo último que hay que hacer es copiar en la carpeta en donde están los archivos del proyecto creado, el archivo del target que tiene extensión .ccxml y que se encuentra en la siguiente dirección C:\Users\“Usuario de la PC”\ti\CCSTargetConfigurations

Para verificar el funcionamiento de la aplicación se desconecta el Piccolo del CCS y se ejecuta el launcher.exe que está dentro de la carpeta del proyecto creado.

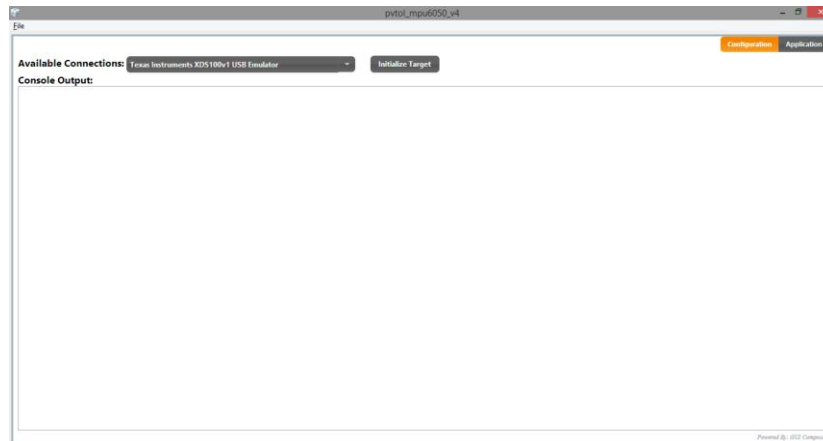


Figura 6.36 Ventana de configuración de la aplicación creada

Al abrirse la aplicación se da clic en el botón *Initialize Target*. Lo que hace este botón es programar al Piccolo y comenzar la aplicación, el tiempo de muestreo está en su valor por defecto y es necesario modificarlo a 100ms.

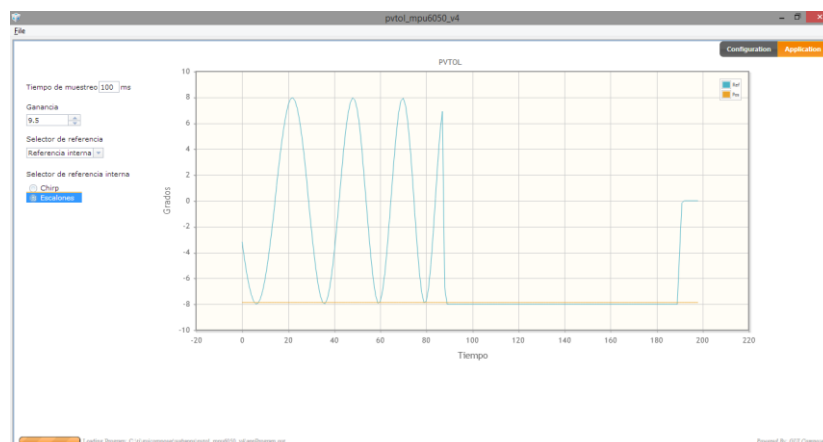
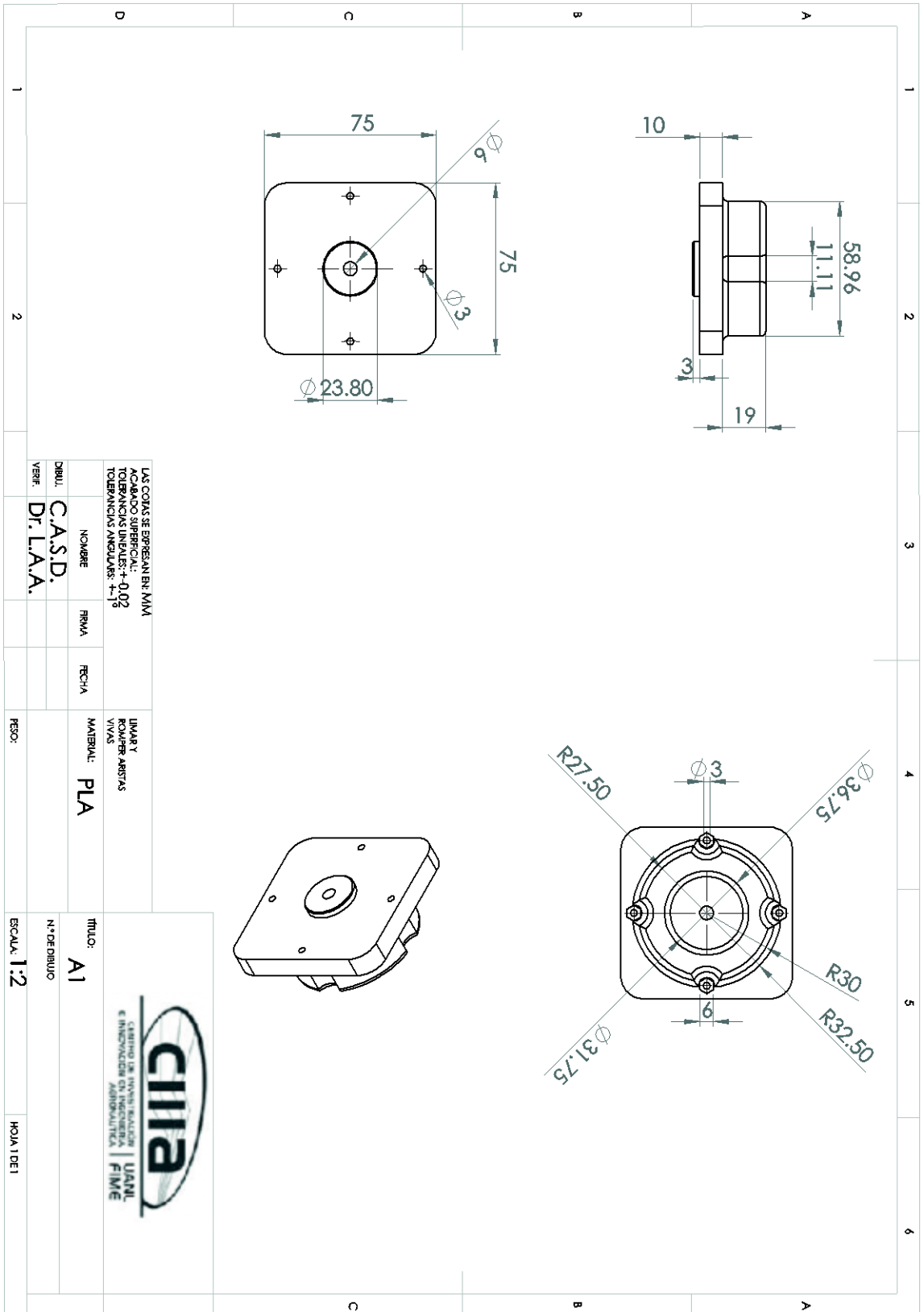
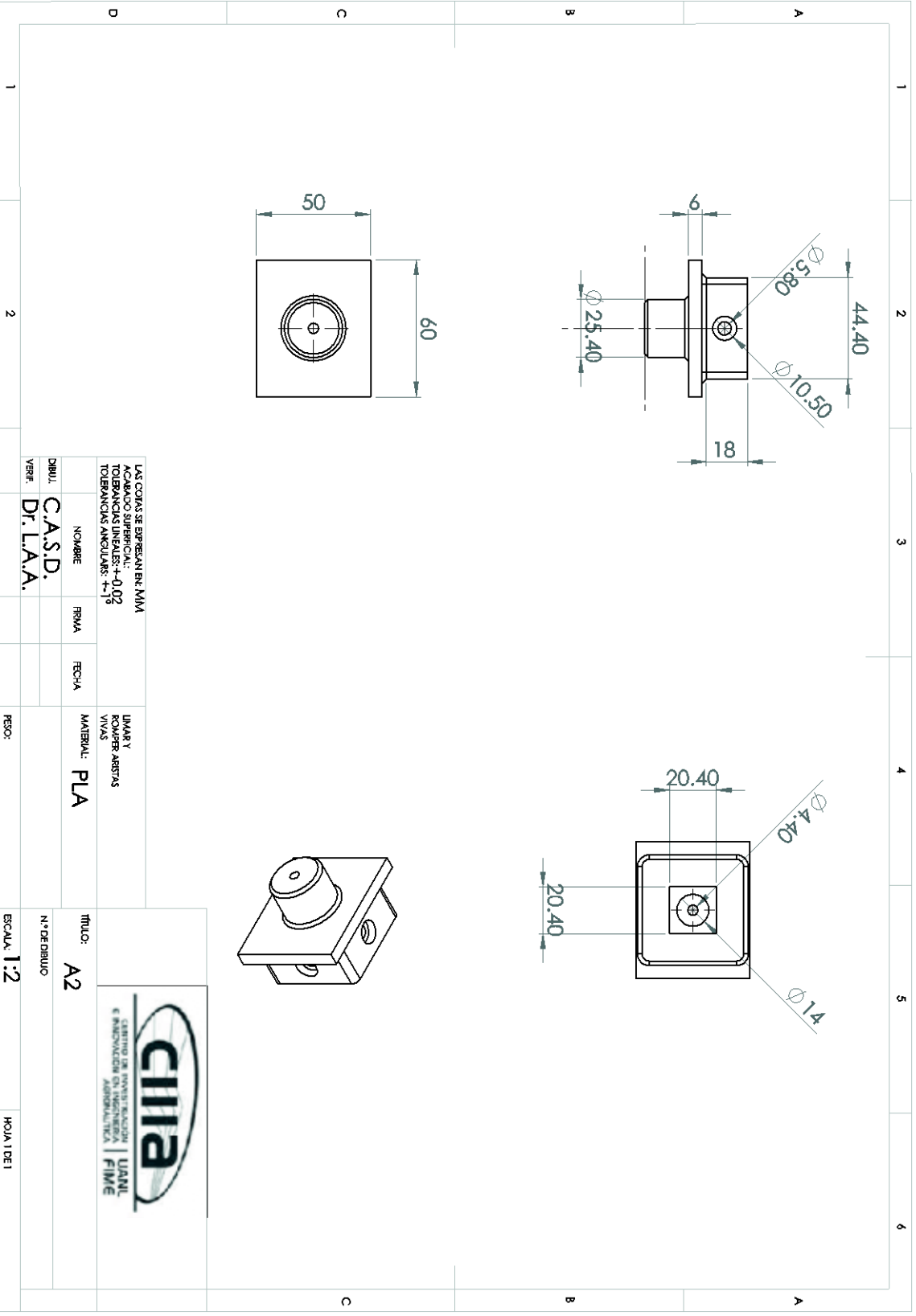


Figura 6.37 Aplicación final en funcionamiento

6.3 Planos





LAS CORNAS SE BARRERAN EN: MM
 TOLERANCIAS LINEALES: ± 0.02
 TOLERANCIAS ANGULARES: $\pm 1^\circ$

UNIDAD Y
 ROJAS ABSTAS
 VIVAS

NOMBRE: **C.A.S.D.**

FNVA

FECHA

MATERIAL: **PLA**

TITULO: **A2**

VERIF.: **DR. LAA.**

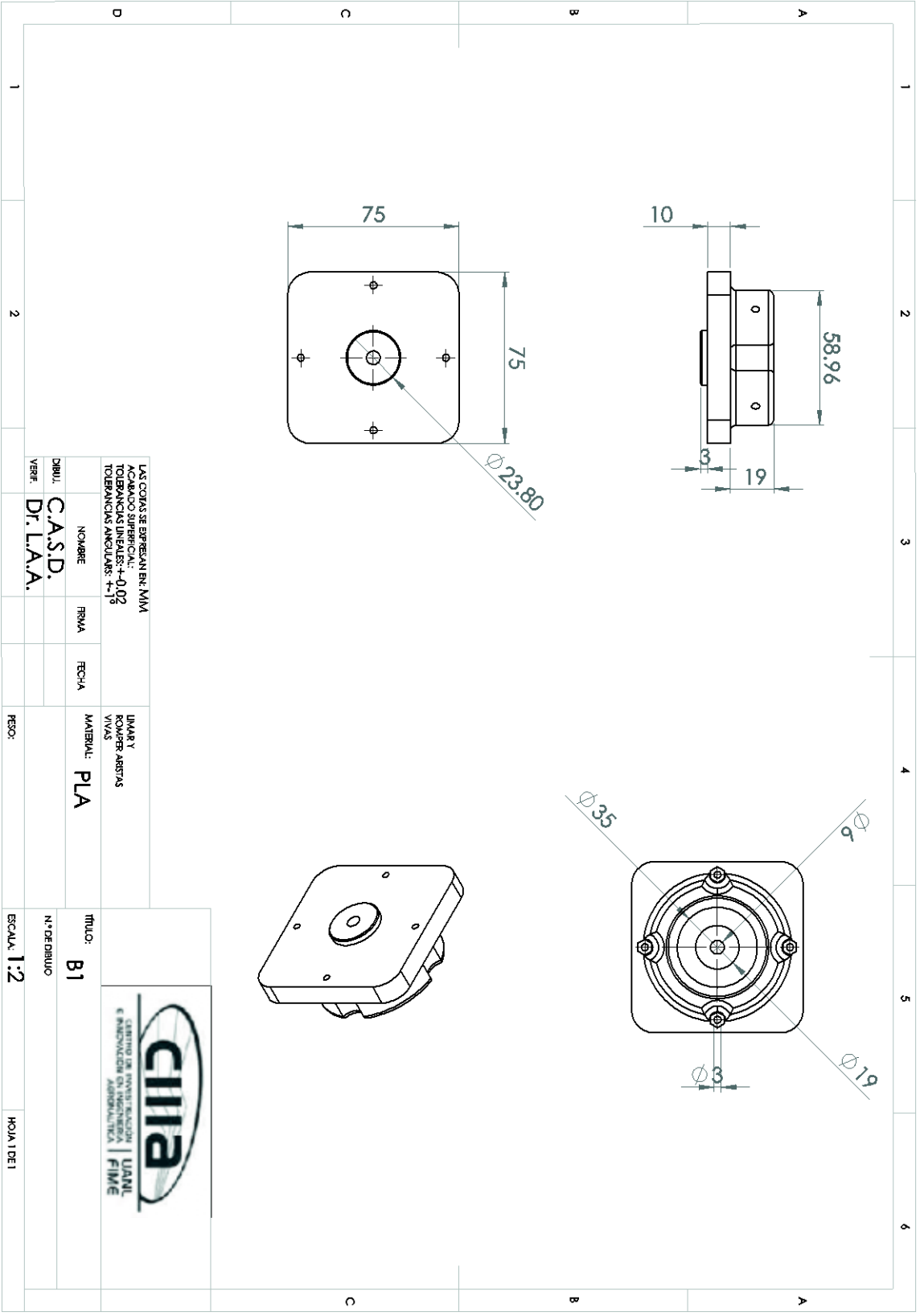
RECO:

N.º DE DIBUJO

ESCALA: **1:2**



HOLA 1 DE 1



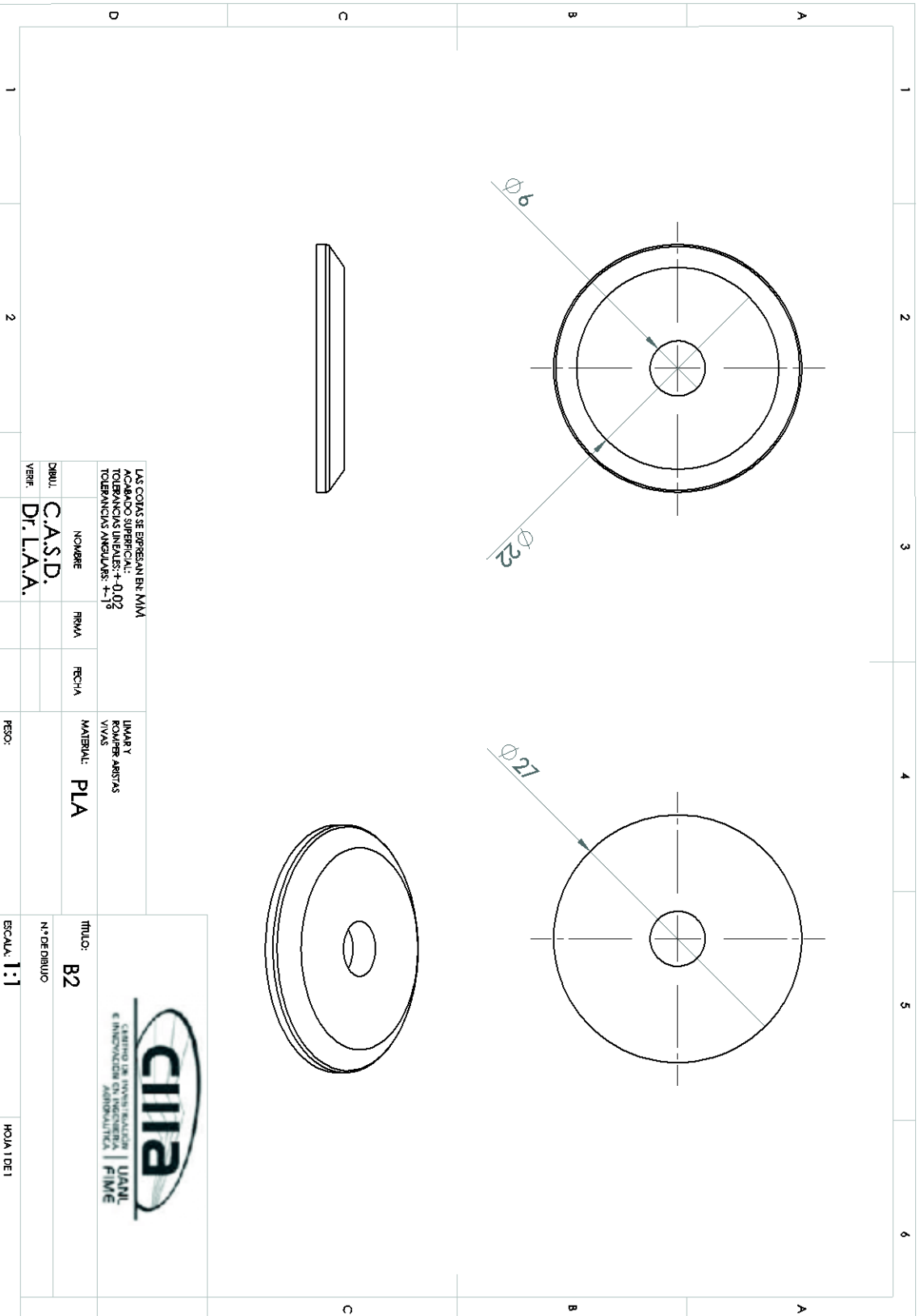
LAS CORNAS SE BARRERAN EN: MM
 TOLERANCIAS LINEALES: ± 0.02
 TOLERANCIAS ANGULARES: $\pm 1^\circ$

UMBAY
 ROAPE ABSTAS
 VIVAS

DIBUJ. VERIF.	NOMBRE C.A.S.D. DR. L.A.A.	RBYA	FECHA	MATERIAL: PLA	PESO:
------------------	--	------	-------	-------------------------	-------

TITULO: B1	N.º DE DIBUJO 1:2	HOJA 1 DE 1
----------------------	-----------------------------	-------------





LAS COTAS SE EXPRESAN EN MM
 TOLERANCIAS LINEALES: ± 0.02
 TOLERANCIAS ANGULARES: $\pm 1'$

IMAGEN
 ROQUE ABSTAS
 VIVAS

TITULO: **B2**

N.º DE DIBUJO

ESCALA: **1:1**

HOLA 1 DE 1

DIBUJ. **C.A.S.D.**
 VERIF. **DR. L.A.A.**

NOMBRE

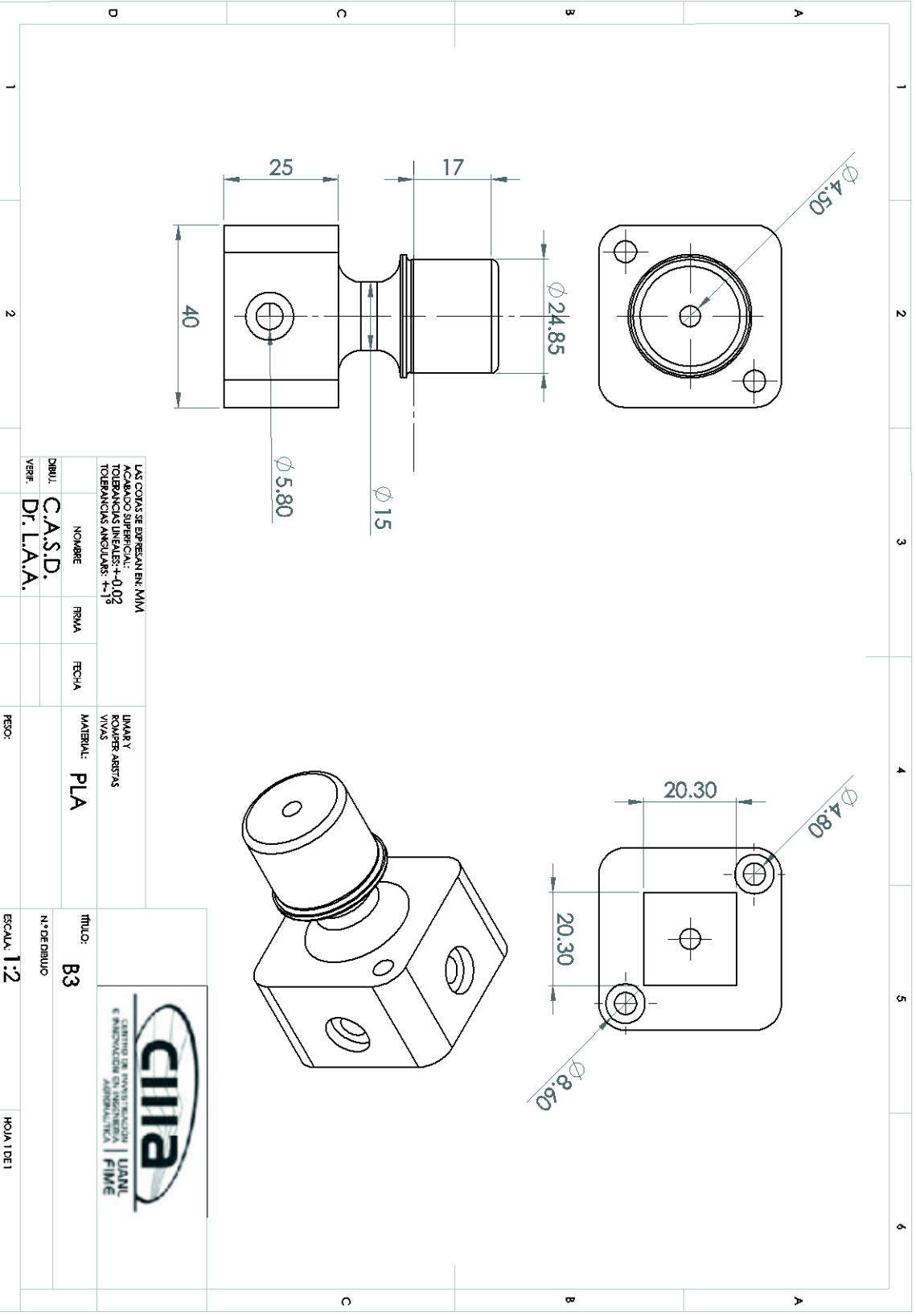
FNVA

FECHA

MATERIAL: **PLA**

RESO:





LAS CORNAS SE BARRERAN EN: MM
 TOLERANCIAS LINEALES: ± 0.02
 TOLERANCIAS ANGULARES: $\pm 1'$

UNIDAD Y
 ROJAS ABSTAS
 VIVAS

DIBUJ. VERIF.	NOMBRE C.A.S.D. DR. L.A.A.	RBYA	FECHA	MATERIAL: PLA	PESO:
------------------	--	------	-------	-------------------------	-------

TITULO: B3	
N.º DE DIBUJO ESCALA: 1:2	
HOJA 1 DE 1	

6.4 Script de Matlab para el cuadri-rotor

```
disp('Cargando variables...')
Ts=0.001;%tiempo de muestreo
disp(' ')
%escalamiento del radiocontrol
%canal 1 alabeo
lmin=1000; lmax=2000;
smin=-1; smax=1;
c=[lmin 1;lmax 1]; d=[smin;smax];
Ks=inv(c)*d;
%canal 2 cabeceo
lmin=1000; lmax=2000;
smin2=-1; smax2=1;
c=[lmin 1;lmax 1]; d=[smin2;smax2];
Ks2=inv(c)*d;
%canal 3 acelerador
lmin=1000; lmax=2000;
smin3=1142; smax3=1850;
c=[lmin 1;lmax 1]; d=[smin3;smax3];
Ks3=inv(c)*d;
%canal 4 guiñada
lmin=1000; lmax=2000;
smin4=-35; smax4=35;
c=[lmin 1;lmax 1]; d=[smin4;smax4];
Ks4=inv(c)*d;
%observador de integrador para alabeo y cabeceo
num=1; den=[1 0];
[A,B,C,D]=tf2ss(num,den); po=(-0.2);
Ht=place(A',C',po); H=Ht';
Ao=A-H*C; Bo=[B H]; Co=1; Do=zeros(1,2);
MFTo=zpk(ss(Ao,Bo,Co,Do));
[ao,bo,co,do]=c2dm(Ao,Bo,Co,Do,Ts,'zoh');
%observador de integrador para guiñada
num2=1; den2=[1 0];
[A2,B2,C2,D2]=tf2ss(num2,den2); po2=(-.8);
Ht2=place(A2',C2',po2); H2=Ht2';
Ao2=A2-H2*C2; Bo2=[B2 H2]; Co2=1; Do2=zeros(1,2);
MFTo2=zpk(ss(Ao2,Bo2,Co2,Do2));
[ao2,bo2,co2,do2]=c2dm(Ao2,Bo2,Co2,Do2,Ts,'zoh');
%controlador de alabeo y cabeceo
disp('Controlador de alabeo y cabeceo')
num=[1 1.75 0.375]; den=[1 60.07 4.2];
c=tf(num,den)
cdrp=c2d(c,Ts);
[ncrp,dcrp]=tfdata(cdrp,'v');
%controlador de guiñada
disp('Controlador de guiñada')
nc3=[1 1.6 0.15]; dc3=[1 60.07 4.2];
cyaw=tf(nc3,dc3)
cdyaw=c2d(cyaw,Ts);
[ncyaw,dcyaw]=tfdata(cdyaw,'v');
disp('Terminado')
% filtro discreto
filtdis=c2d(tf(300*300,poly([-300 -300])),Ts);
[nfd,dfd]=tfdata(filtdis,'v');
```