UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

“Compensación Mecánica para la Estabilización de un Sistema de Procesamiento Visual”

TESIS

Que para obtener el grado de:

Ingeniero Mecatrónico

Presenta:

Edwinn Gamborino Castañeda

Director de tesis:

Dr. Victor Javier González Villela

Ciudad Universitaria, México, D.F. **Mayo 2013**

AGRADECIMIENTOS

Quiero agradecer a todas las personas que hicieron posible el desarrollo y consumación de este proyecto. Especialmente a mis padres, Antonio Gamborino y Elvira Castañeda. Ha sido un largo camino y no pude haberlo logrado sin su apoyo.

Thanks to all the members of Yagi Laboratory (BioSystems and Devices Area, Division of Electrical, Electronic and Information Engineering, Graduate School of Engineering, Osaka University) for helping me throughout this project whenever I needed. Thanks to Osaka University for allowing me to use its installations as I needed them.

RESUMEN

Aunque han habido varios intentos desarrollados en años recientes para proveer de estabilización activa a los sistemas de visión mediante diferentes métodos, todos han utilizado matemáticas avanzadas, tales como operaciones matriciales, cinemática inversa, control robusto, entre otros; estos tipos de métodos requieren una gran capacidad de procesamiento computacional para funcionar correctamente. El enfoque que se tuvo en este proyecto fue el de disminuir la carga operativa para poder operar el sistema utilizando exclusivamente sistemas embebidos, por lo tanto la simplicidad fue uno de los puntos principales del diseño.

Este documento provee una introducción detallada al principio de operación de los sensores inerciales giroscopio y acelerómetro, asi como a los sistemas inteligentes de visión y a la teoría de control discreto. Todos estos fundamentos se combinan para resolver el problema de estabilización de un sistema de visión por medios mecánicos utilizando sensores que poseen diferentes tiempos de muestreo. Un proyecto se realizó a partir de este documento y los resultados de los experimentos realizados a dicha plataforma serán presentados al final del mismo.

CONTENIDOS

ÍNDICE DE FIGURAS

Capítulo 3.

Capítulo 4.

ÍNDICE DE TABLAS

CAPÍTULO 1

INTRODUCCIÓN

1. OBJETIVOS DEL PROYECTO

1.1. Objetivo principal

Diseñar y construir una sistema de estabilización en un grado de libertad el cual, utilizando los datos entregados por diferentes sensores inerciales y visuales, compense mecánicamente el movimiento inducido por perturbaciones externas en una plataforma móvil.

1.2. Sub-objetivos

- Caracterizar los componentes seleccionados realizandor pruebas de desempeño a los mismos.
- Diseñar un algoritmo que permita combinar los datos de los sensores utilizados tomando en cuenta las propiededades de los mismos, tales como velocidad de muestreo, rango, sensibilidad, entre otros.
- Construir un prototipo funcional experimental de acuerdo a los requerimientos del proyecto.

2. DESCRIPCIÓN DEL PROYECTO

La estabilización activa de los sistemas de visión robótica es de vital importancia para el uso de cámaras tele-operadas de alta resolución en vehículos autónomos. Pequeños ángulos de apertura, aunados a grandes longitudes focales provocan una alta sensibilidad al movimiento rotacional generado por el vehículo en movimiento, por ejemplo, por golpes o frenado. Debido a las grandes latencias (retardos temporales en la transmisión de datos) presentes en los sistemas de procesamiento visual que utilizan solamente la información provista por la cámara, el tiempo de reacción puede no ser suficientemente rápido para contrarrestar los movimientos súbitos previamente descritos. Por lo tanto, existe una necesidad de implementar un sensor que nos provea de información acerca del movimiento del sistema, pero que además posea un tiempo de muestreo mucho más rápido, tal es el caso de los sensores inerciales. En particular se seleccionaron un sensor de momento angular (giroscopio) y un sensor de aceleración lineal (acelerómetro).

Aunque han habido numerosos intentos en los últimos años para lograr la estabilización visual activa a través de diferentes métodos, el enfoque adoptado por el proyecto siguiente es el de desarrollar un algoritmo con una baja carga computacional que pueda ser realizado por sistemas embebidos, por lo tanto, la simplicidad del diseño se mantuvo siempre en cuenta para el diseño de esta plataforma.

El siguiente trabajo presenta un algoritmo que ha demostrado experimentalmente combinar con éxito los datos proporcionados por los dos sensores y la retroalimentación visual, teniendo en cuenta las latencias de muestreo de cada sensor y las capacidades de procesamiento de los sistemas embebidos.

La información se presenta de la siguiente manera:

- En el Capítulo 1 se presenta una breve introducción al proyecto, incluyendo la definición de los objetivos, la problemática y el estado del arte en el campo.

- En el Capítulo 2 se da una descripción de los sensores inerciales, incluyendo su principio de operación, una breve descripción de los mismos y los protocolos de comunicación requeridos por los circuitos integrados específicos utilizados para las pruebas experimentales. También en la Sección II se provee una descripción del sensor visual. El sensor utilizado en la experimentación actual es la *Silicon Retina*, desarrollado por Tetsuya et al.[12] Este sensor es controlado por un FPGA *Spartan III*, el cual también es discutido brevemente en esta sección. También se discute brevemente en esta sección las características y capacidades de las plataformas PSoC 1 de *Cypress Semiconductor* y Arduino UNO utilizadas en el sistema de control desarrollado.

- En el Capítulo 3 se encuentra una descripción de la plataforma de experimentación construida, incluyendo características mecánicas, eléctricas y electrónicas, asi como una descripción del procesamiento de imagen utilizado y el *logger*, sistema utilizado para guardar los datos de los sensores. Además se presenta una descripción detallada de la transmisión de información entre dispositivos.

- En el Capítulo 4 se describen los diagramas de control desarrollados utilizando los sensores de inercia y la retroalimentación visual, en primera instancia cada uno separado y posteriormente combinados. Esta descripción incluirá una introducción a la teoría de los sistemas de control discreto y a la interfaz de programación de la plataforma.

- El documento cierra con una descripción de los experimentos realizados, los resultados obtenidos a partir de dichos experimentos y se resaltan algunas conclusiones. Las posibilidades de futuro desarrollo de este proyecto también se discuten al final de esta sección.

- Posteriormente se encuentran en los anexos el articulo original redactado en la Universidad de Osaka, Japón, y los códigos completos utilizados para las plataformas PSoC y Arduino como medio de consulta.

3. ANTECEDENTES

3.1. Justificación

Aunque han habido varios intentos desarrollados en años recientes para proveer de estabilización dinámica a los sistemas de visión mediante diferentes métodos [2][3], todos han utilizado matemáticas avanzadas, tales como operaciones matriciales, cinemática inversa, control robusto, entre otros. Este tipo de métodos requieren una gran capacidad de cómputo para funcionar correctamente. El enfoque que se tuvo en este proyecto fue tomando en cuenta disminuir la carga operativa en sistemas embebidos, por lo tanto la simplicidad fue uno de los puntos principales del diseño.

3.2. Formulación del Estado del Arte

En años recientes han habido varios enfoques para resolver el problema de la estabilización de los sistemas de visión. A continuación se presentan tres ejemplos sobresalientes que utilizan diferentes métodos para resolver el problema:

1. **Cinemática inversa (Rochester University, 1991)** [4]

La problemática presentada en este proyecto fue la implementación de un sensor de visión en el actuador final de un brazo robótico de seis grados de libertad. Para resolverlo se utilizó el método de cinemática inversa para alimentar al sistema de control que manipula a dos cámaras paralelas. Este método requiere álgebra matricial por lo tanto es imposible realizarlo en tiempo real sin un procesador de capacidad considerable. A continuación se presentan algunas figuras del diseño desarrollado (Figura 1.1).

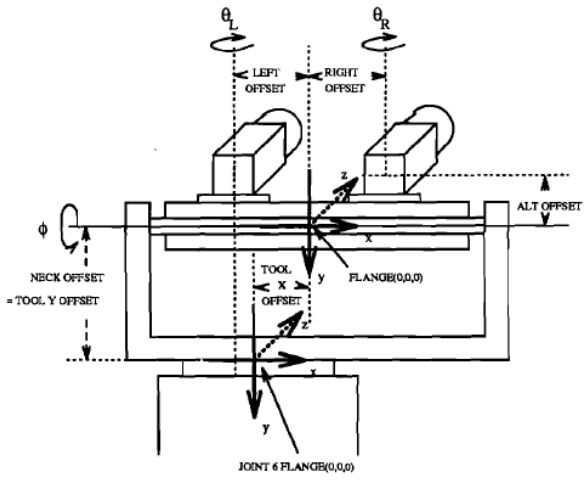


Fig. 1.1: Esquema 3D del sistema estereoscópico [4]

2. Redes neuronales (Japón, 2001) [5]

Este modelo fue desarrollado por una empresa privada en Japón. El sistema utilizó redes neuronales para que una cabeza robótica con tres grados de libertad pudiera aprender por sí misma cómo reaccionar a diferentes estímulos captados por el sensor de visión. De nuevo, el proceso de redes neuronales requiere una gran cantidad de programación y una fuerte capacidad de procesamiento para poder llevarse a cabo, sin embargo, el enfoque de auto aprendizaje tiene una capacidad considerable y podría llegar a aplicarse a una extensión de este proyecto en un momento dado (Figuras 1.2 y 1.3).

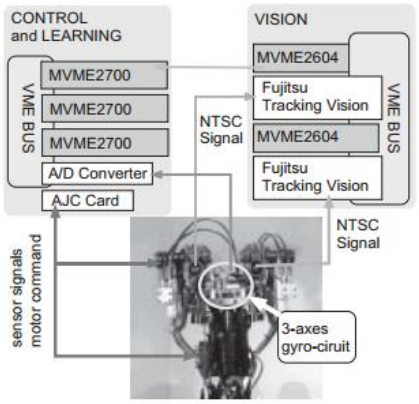


Fig. 1.2: Mapa conceptual del diseño [5]

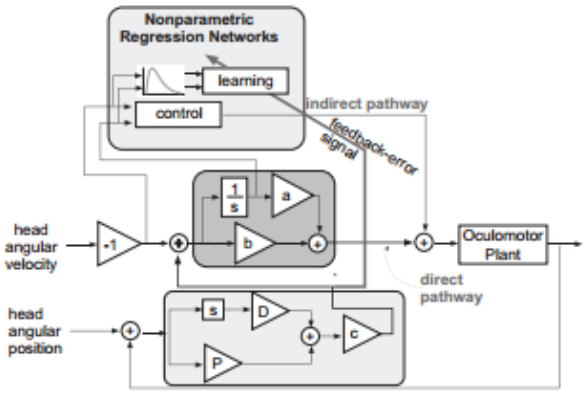


Fig. 1.3: Diagrama de control correspondiente [5]

3. Control Adaptable (MRAC) (Tohoku University, 2005) [6]

Este sistema fue desarrollado para un robot humanoide utilizando un sistema de control adaptativo (MRAC, inglés: *Model Referenced Adaptative Control*) para procesar la información. Un control adaptable es similar a un control tradicional de lazo cerrado en el cual las constantes del controlador pueden cambiar dinámicamente de acuerdo a las lecturas de un sensor, en este caso un sensor de presión puesto en la base de la planta del pie del robot, lo cual cambia en términos del tipo de terreno que se pisa. Para este modelo se requieren modelos matemáticos como *Filtros de Kalman* o *Lagrangianos*, nuevamente, gran capacidad de procesamiento es requerida (Figuras 1.4 y 1.5).

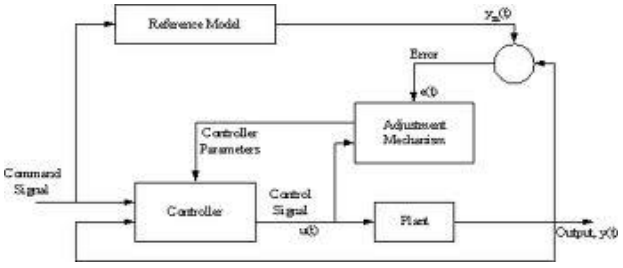


Fig. 1.4: Diagrama básico de un control adaptivo[6]

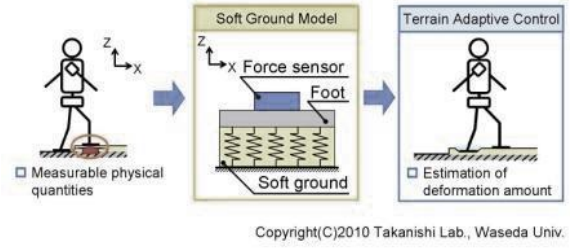


Fig. 1.5: Sensor que regula al controlador activamente[6]

3.3. Marco Conceptual

El proyecto está definido por los siguientes principios básicos:

- **Adaptabilidad a diferentes sistemas:** Se busca que esta plataforma pueda ser adaptada a cualquier tipo de sistema móvil, desde un automóvil, un brazo robótico, un robot humanoide, hasta un sistema satelital.

- **Interfaz de comunicación veloz y sencilla:** Un objetivo final del módulo será que pueda enviar los datos al procesador del sistema que lo ocupe de manera rápida y confiable, mediante un protocolo fácil de entender y de sencilla implementación.

- **Simplicidad de sistemas** y mecanismos para mantener los costos relativamente bajos.

- **Capacidad del operador de conocer todo el funcionamiento interno** del módulo, tanto en hardware como en software y la capacidad de modificarlo a voluntad para adaptarlo con la mayor flexibilidad posible a cualquier proyecto.

- **La calibración del sistema se realizará por el operador** y no de forma continua-automática como es el caso de los costosos sistemas actuales. Con ello se podrá simplificar el sistema y permitir un costo mucho menor al no requerir sistemas avanzados de control dinámico.

CAPÍTULO 2

DESCRIPCIÓN DE LOS COMPONENTES

1. ACELERÓMETRO

1.1. Introducción

Aceleración es la medida de qué tan rápido cambia la velocidad de un cuerpo dado. Esto no es necesariamente la misma que la aceleración de coordenadas (cambio de la velocidad del dispositivo en el espacio), sino que es el tipo de aceleración asociada con el fenómeno de peso experimentada por una masa de prueba que se encuentra en el marco de referencia del dispositivo por efecto de la gravedad.

Tal como un velocímetro mide velocidad, un acelerómetro mide aceleración. Los acelerómetros pueden utilizarse en una variedad de aplicaciones que son útiles para diseños y proyectos electrónicos y robóticos, tales como: medir aceleración, inclinación o ángulo de inclinación (referidos al vector de aceleración de la gravedad), vibración, gravedad e impactos, entre otros.

1.2. Principio de Funcionamiento

2. Derivación de la Ecuación de Movimiento [8][9]

La medición de la aceleración siempre se basa en mecánica clásica Newtoniana. Normalmente la aceleración a la cual un cuerpo está sujeta es de interés para la formulación de modelos. Los transductores que son utilizados en el documento presente pueden ser modelados como un elemento sísmico consistente de una masa de prueba suspendida por un resorte; la aceleración genera una fuerza sobre la masa la cual es consecuentemente reflejada en una elongación x del resorte, como se muestra en la Figura 2.1. Alguna forma de amortiguamiento es requerida, de otro modo idealmente la masa oscilaría a su frecuencia natural 'f_n' para cualquier señal de entrada.

Es de notarse que los acelerómetros modernos no utilizan masas suspendidas para medir la aceleración, si no que se basan en diferentes técnicas desarrolladas utilizando MEMS (Sistemas micro-electromecánicos, inglés: *MicroElectroMechanical Systems*).

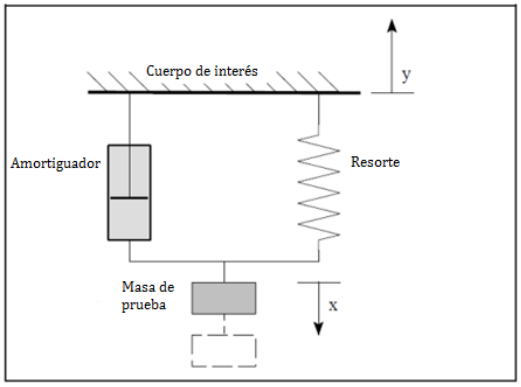


Fig. 2.1: Principio de operación del elemento sensitivo de un acelerómetro.[9]

Para derivar la ecuación de movimiento del sistema, se aplica el Principio de *D'Alembert*, donde todas las fuerzas actuando sobre la masa de prueba son iguales a la fuerza de inercia sobre la misma. Desde un punto de vista estacionario, la suma de todas las fuerzas en la dirección *y* es:

$$m\frac{d^2(x(t)-y(t))}{dt^2}+\mathrm{b}\frac{dx(t)}{dt}+kx(t)=0$$

$$m\frac{d^2y(t)}{dt^2}=m\frac{d^2x(t)}{dt^2}+b\frac{dx(t)}{dt}+kx(t) \qquad [2.1]$$

Donde:

m: masa de la masa de prueba
y: movimiento del cuerpo de interés
x: movimiento de la masa de prueba
b: factor de amortiguamiento
k: constante del resorte

De la Ecuación [2.1] podemos obtener la respuesta tanto para el estado transitorio como el permanente como se demuestra en las secciones 2 y 3 a continuación.

3. Desempeño ó respuesta en estado permanente:

En estado permanente, esto es, con aceleración constante *a*, y elongación constante *x*, la Ecuación [2.1] entrega:

$$ma=kx \qquad [2.2]$$

$$a = \frac{kx}{m}$$

De la Ecuación [2.2] podemos definir un parámetro llamado sensiblidad, el cual describe una relación entre la elongación del resorte x y una aceleración a correspondiente, o bien entre la masa m y la constante del resorte k, tal como se muestra en la Ecuación [2.3]:

$$s = \frac{x}{a} = \frac{m}{k} \qquad [2.3]$$

4. Desempeño ó respuesta en estado transitorio:

Para analizar el estado transitorio es más sencillo utilizar la Transformada de Laplace de la Ecuación [2.1]:

$$\frac{x(s)}{a(s)} = \frac{1}{s^2 + \frac{b}{m}s + \frac{k}{m}}$$

De esta ecuación, la frecuencia natural f_n es:

$$f_n = \sqrt{\frac{k}{m}} \qquad [2.4]$$

La frecuencia natural fn nos permite conocer la resonancia del sistema y así definir otros parámetros como el factor de amortiguamiento para caracterízar al sistema. Una vez caracterizado es posible conocer las propiedades de la respuesta transitoria, tales como el tiempo de asentamiento y el sobrepaso.

Mientras que la Ecuación [2.1] modela el comportamiento de un acelerómetro; aunque parámetros tales como la sensibilidad y la frecuencia natural de un acelerómetro comercial son especificadas ya en la hoja de datos, de no contar con dichos elementos se pueden calcular utilizando las Ecuaciones [2.2] a [2.4], caracterizando así a nuestro sensor.

1.3. Acelerómetro analógico MMA7361L [10]

1. Descripción:

El MMA7361L es un acelerómetro analógico de bajo consumo, baja capacitancia, micromaquinado con características tales como acondicionamiento de señales, un filtro pasa bajas de un polo, compensación por temperatura, auto pruebas, *0g-Detect* el cual detecta caída libre lineal y *g-Select* el cual permite seleccionar entre dos sensibilidades diferentes.

El dispositivo consiste de una celda capacitiva de superficie micromaquinada (Véase Figura 2.2), también llamada *g-cell*, y un modulo de acondicionamiento de señales ASIC contenidos en un solo empaque. El elemento sensor esta sellado herméticamente utilizando una capa de material aislante alrededor del mismo.

La *g-cell* es una estructura mecánica conformada con materiales semiconductores (polisilicón) utilizando procesos para semiconductores (enmascaramiento y grabado). Puede ser modelado como un conjunto de varillas unidas a una masa central que se mueve entre varillas fijas. Las varillas móviles pueden cambiar de posición haciendo que el sistema cambie su aceleración.

Conforme las varillas pegadas a la masa central se mueven, la distancia desde estas a las varillas fijas de un lado aumentará la misma distancia que disminuye del otro lado. Este cambio de distancia es una medida de aceleración. Este cambio es detectado por los electrodos capacitivos montados en las varillas fijas, convertido a un voltaje que puede ser interpretado como un valor proporcional a la aceleración del sistema. Véase la figura 2.2 debajo.

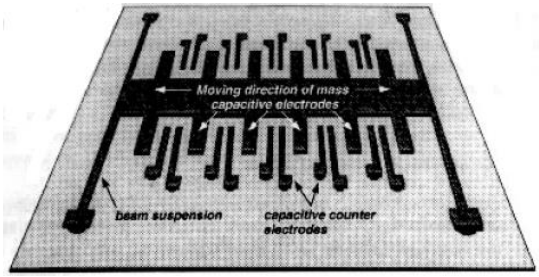


Fig. 2.2: Acelerómetro micromaquinado de silicón. [8]

2. Características eléctricas y *pinout*:

A continuación se presentan las características generales del sensor seleccionado. Nótese que la sensibilidad es amplia para un sensor analógico y al trabajar con voltajes menores a 3.3 V su consumo de energía es mínimo. En las Figura 2.4 se observa la orientación axial del chip físicamente.

Parámetro	Típico	Limite	Unidad
Voltaje de alimentación	3.3	2.2 a 3.6	V
Corriente de alimentación	400	Hasta 600	μA
Corriente en modo de ahorro	3	10	μA
Temperatura operacional	-	-40 a 85	°C
Salida en gravedad cero	1.65	1.485 a 1.815	V
Sensibilidad	800	740 a 860	mV/g
Sensibilidad	206	190 a 221	mV/g
Impedancia	32	-	KΩ

Tabla 2.1: Características operativas del MMA7361L. [10]

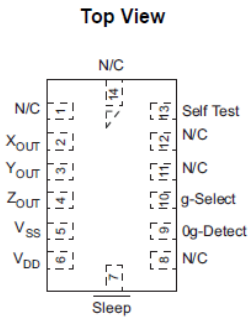


Fig. 2.3: Descripción de los pines.

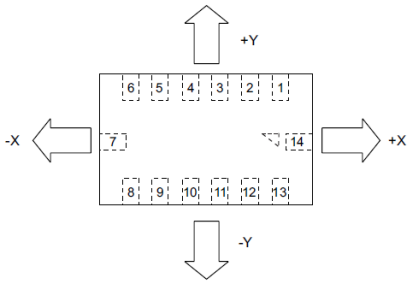


Fig. 2.4: Orientación axial del MMA7361L.

1.4. Convertidor Analógico Digital (ADC) [11]

Un convertidor analógico digital (comúnmente conocido como ADC, A/D, entre otros) es un dispositivo electrónico que convierte una entrada de voltaje o corriente analógica (y continua) a un numero digital (y discreto) proporcional a la magnitud de dicho voltaje o corriente en términos de dos referencias (Voltaje de polarización y tierra). Los dispositivos digitales, tales como microcontroladores, suelen utilizar los convertidores ADC para leer los voltajes de entrada de diferentes sensores analógicos.

Para ADCs de voltaje, existen tres tipos de estructura diferentes: De un polo, Pseudo-Diferencial, y Completamente Diferencial. Cada tipo tiene sus ventajas y desventajas y la topología debe ser seleccionada de acuerdo al tipo de señal que debe ser discretizada. Una breve descripción de cada topología se da en las siguientes subsecciones:

1. ADC completamente diferencial

La Figura 2.5 muestra un ejemplo de un ADC completamente diferencial con una estructura de entrada de tipo T/H (Inglés: *Track and Hold*, Español: Muestreo y Espera) Durante el modo de muestreo, $C_{sample(+)}$ carga a $[A_{IN(+)}$ Ð $V_{DD}/2]$ y $C_{sample(-)}$ carga a $[A_{IN(-)}$ Ð $V_{DD}/2]$ (Véase Figura 2.5). Cuando los interruptores T/H cambian a modo de espera, $Cs_{ample(+)}$ y $C_{sample(-)}$ se conectan en serie de tal forma que el voltaje de muestreo presentado al ADC es la diferencia entre $A_{IN(+)}$ y $A_{IN(-)}$. La arquitectura diferencial en conjunto con un ancho de banda de entrada aceptable en el T/H son ingredientes clave para una buena dinámica del sistema.

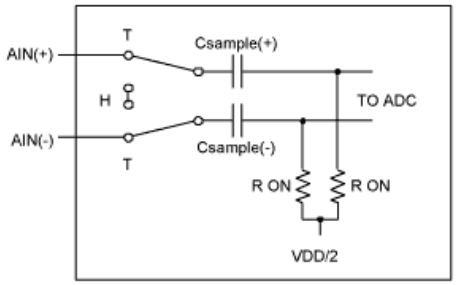


Fig. 2.5: Circuito T/H completamente diferencial.[11]

2. ADC Pseudo-Diferencial

Las entradas pseudo-diferenciales son similares a las completamente diferenciales en cuanto a que separan la tierra del circuito ADC de la tierra de la señal de entrada, permitiendo que los voltajes comunes de DC se cancelen entre si (efecto que los ADC de un polo no poseen). Sin embargo, a diferencia de las entradas completamente diferenciales, éstas tienen muy poco efecto en el ruido dinámico. En la Figura 2.6, el muestreo solo ocurre en la señal $A_{IN(+)}$. La señal común, $A_{IN(-)}$, no es muestreada. Durante el modo de muestreo, el capacitor se carga a través del resistor en serie R_{ON}. Durante el modo de espera, el capacitor se conecta a $A_{IN(-)}$ y una señal invertida es presentada al ADC para convertir. Debido a que el muestreo solo ocurre en la entrada positiva, la entrada negativa debe permanecer en un rango de ±0.1LSB durante la conversión para un desempeño óptimo.

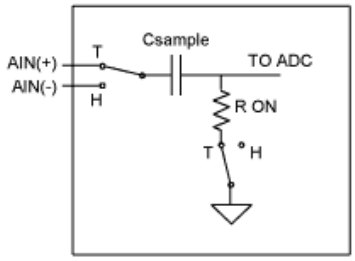


Fig. 2.6: Circuito T/H Pseudo-Diferencial.[11]

3. ADC de un polo

Las entradas de un solo polo por lo general son suficientes para la mayoría de las aplicaciones. En aplicaciones de un solo polo, todas las señales son referidas a la tierra común en el circuito ADC. Cada canal utiliza un solo pin de entrada, la tierra analógica es compartida entre todas las entradas para los sistemas multi-canal. El offset de DC y/o el ruido en el camino de la señal causan un decremento del rango dinámico de la señal de entrada. Las entradas de un solo polo son ideales cuando la fuente de la señal y el ADC están cerca el uno del otro (por ejemplo, en la misma placa de manera que los caminos de material conductor entre uno y otro puedan mantenerse tan cortos como sea posible). Las entradas de un polo son más susceptibles al ruido de acoplamientos y offsets de DC, sin embargo, utilizando circuitos analógicos de acondicionamiento de señales estos efectos pueden amortiguarse

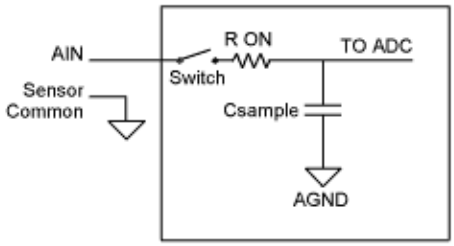


Fig. 2.7: Circuito T/H de un solo polo.[11]

La mayoría de los microcontroladores con capacidad de realizar ADC dentro de un mismo microchip utilizan la topología de un solo polo, debido a su simplicidad de diseño y uso.

2. GIROSCOPIO

2.1. Introducción

Un giroscopio es un dispositivo utilizado para medir o mantener cierta orientación, basado en los principios de momento angular. Está formado esencialmente por un cuerpo con simetría de rotación que gira alrededor del eje de dicha simetría. Cuando el giroscopio se somete a un momento de fuerza que tiende a cambiar la orientación de su eje de rotación, tiene un comportamiento aparentemente paradójico, ya que cambia de orientación (o experimenta un momento angular en todo caso, si está restringido) girando respecto de un tercer eje, perpendicular tanto a aquel respecto del cual se lo ha empujado a girar, como a su eje de rotación inicial. Si está montado sobre un soporte de Cardano que minimiza cualquier momento angular externo, o si simplemente gira libre en el espacio, el giróscopo conserva la orientación de su eje de rotación ante fuerzas externas que tiendan a desviarlo mejor que un objeto no giratorio; se desvía mucho menos, y en una dirección diferente.

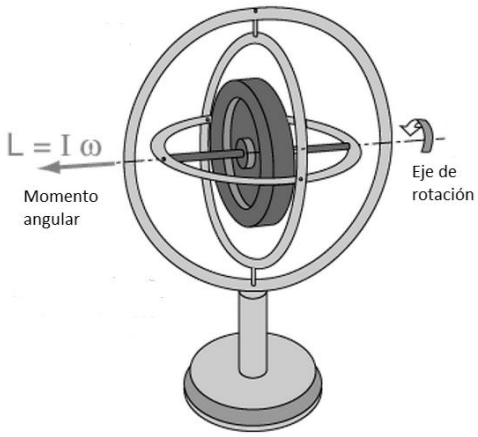


Fig. 2.8: Giroscopio mecánico de un eje montado sobre un soporte de Cardano

Un giroscopio presenta, por tanto, dos propiedades fundamentales: la inercia giroscópica o "rigidez en el espacio" y la precesión, que es la inclinación del eje en ángulo recto ante cualquier fuerza que tienda a cambiar el plano de rotación (Véase Figura 2.9). Estas propiedades se manifiestan a todos los cuerpos en rotación, incluida la Tierra. El término giroscopio se aplica generalmente a objetos esféricos o en forma de disco montados sobre un eje, de forma que puedan girar libremente en cualquier dirección; estos instrumentos se emplean para demostrar las propiedades anteriores, para indicar movimientos en el espacio, o para producirlos.

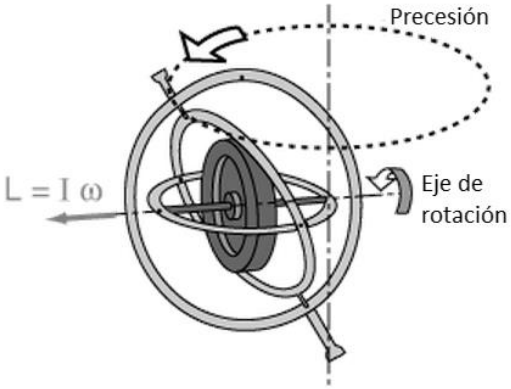


Fig. 2.9: Efecto del vector de inercia sobre la estabilidad del cuerpo giratorio, generando precesión.

Éste fenómeno físico, el efecto giroscópico, puede observarse fácil y cotidianamente en juguetes como trompos o pirinolas o monedas lanzadas a rodar, por ejemplo, aunque por supuesto, cualquier objeto giratorio funciona en cierto modo, como giroscopio. El giro en vuelo impartido por el jugador a un balón de rugby, o el de una bala disparada desde un arma de ánima rayada para estabilizar su trayectoria son ejemplos de aplicación del efecto.

2.2. Principio de Operación

1. Giroscopio mecánico [12]

Mecánicamente, un giroscopio es una rueda o disco giratorio en el que el eje de rotación es libre de moverse en cualquier dirección. Aunque esta orientación no permanece completamente fija, cambia en respuesta a un torque externo en mucho menor medida (y en una dirección diferente) de lo que cambiaría sin el enorme momento angular asociado con la velocidad de giro del disco y su momento de inercia. Dado que el torque externo es minimizado montando el dispositivo en gimbales, su orientación permanece casi estática, sin importar algún movimiento de la plataforma sobre la cual el dispositivo está montada.

La ecuación fundamental para describir el comportamiento del giroscopio es:

$$\bar{\tau} = \frac{d\bar{L}}{dt} = \frac{d(l\bar{\omega})}{dt} = \mathrm{I}\bar{\alpha} \qquad [2.5]$$

donde los vectores τ y L son, respectivamente, el torque del giroscopio y su momento angular, el escalar I es su momento de inercia, el vector ω es su velocidad angular y el vector α es su aceleración angular.

De la Ecuación [2.5] podemos concluir que al aplicar un torque τ aplicado perpendicularmente al eje de rotación, y por lo tanto perpendicular a *L*, resulta una rotación alrededor de un eje perpendicular a ambos τ y *L*. Este movimiento se conoce como precesión. La velocidad angular de la precesión Ω_P está dado por el producto cruz:

$$\bar{\tau} = \bar{\Omega}_p \; x \; \bar{L}$$

Al ser sometido a un torque constante de magnitud τ, la velocidad de precesión del giroscopio Ω_P es inversamente proporcional a L, la magnitud de su momento angular:

$$\tau = \Omega_p L sin(\theta) \qquad [2.7]$$

Donde θ es el ángulo entre los vectores Ω_P y L. Por lo tanto, si la velocidad angular del giroscopio se reduce (por ejemplo, debido a la fricción), su momento angular disminuirá y en el mismo grado aumentará la precesión. Esto continuará hasta que el dispositivo sea incapaz de girar a una velocidad suficiente para soportar su propio peso, cuando deja de preceder y caerá de su eje de soporte. Por convención estos tres vectores - torque, giro y precesión - están orientados unos con respecto a otros mediante la regla de la mano derecha, es decir, son vectores ortogonales.

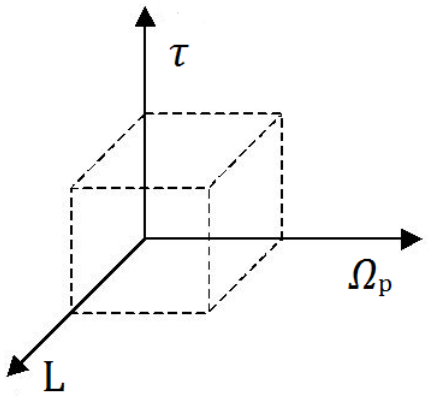


Fig. 2.10: Vectores ortogonales en un sistema giroscópico.

Mientras que entender el principio de operación de un giroscopio mecánico es importante para explicar sus aplicaciones hoy en día, los dispositivos que utilizan estas tecnologías en los campos de electrónica de consumo, robótica y ingeniería aeroespacial utilizan sensores electrónicos que funcionan con principios completamente distintos pero que sin embargo, entregan la misma información. A continuación se explica el funcionamiento de un giroscopio electrónico actual.

2. Giroscopio de estructura vibratoria [13]

Un giroscopio de estructura vibratoria es un tipo de giroscopio que funciona de manera muy similar a los halterios de un insecto (estructuras biológicas que le dan orientación espacial a los insectos voladores, Figura 2.11). El principio físico detrás de ambos casos consiste en que un objeto vibratorio tiende a continuar vibrando en un mismo plano mientras su soporte se encuentra rotando. En la literatura ingenieril, este tipo de dispositivo también se conoce como un giroscopio vibratorio de Coriolis porque conforme su plano de oscilación gira, la respuesta detectada por el transductor resulta de la ecuación de movimiento de Coriolis.

Fig. 2.11: Halterios de un insecto.

Considere dos masas de prueba vibrando en un plano (tal como en un giroscopio MEMS) a una frecuencia de oscilación ω_r. Recordando que el efecto de Coriolis induce una aceleración en la masas de prueba igual a $a_c = -2(vx\Omega)$, donde *v* es la velocidad lineal y Ω es la velocidad angular de rotación. La velocidad dentro de un solo plano de las masas es dado por $x_{ip}\omega_r\cos(\omega_r t)$, mientras que la posición de las mismas está dada por $x_{ip}sin(\omega_r t)$. El movimiento fuera del plano y_{op}, inducida por la rotación, está dada por:

$$y_{op} = \frac{F_c}{k_{op}} = \frac{2m\Omega x_{ip}\omega_r\cos(\omega_r t)}{k_{op}} \quad [2.8]$$

Donde:

- *m* es la masa de la masa de prueba
- k_{op} es la constante de Young del resorte en una dirección ortogonal al plano de movimiento.
- Ω es la magnitud del vector de rotación dentro del plano y perpendicular a la masa de prueba.

Este tipo de giroscopios electrónicos vienen en paquetes similares a otros circuitos integrados y pueden proveer de salidas analógicas o digitales. En muchos casos, un solo componente incluye giroscopios para múltiples ejes. Algunos componentes pueden incorporar inclusive un acelerómetro y un giroscopio en el mismo circuito, en cuyo caso la salida es un vector de seis grados de libertad.

Internamente, los giroscopios fabricados con tecnología MEMS usan versiones construidas litográficamente de mecanismos tales como anillos vibratorios, tenedores de sincronización o sólidos resonantes de varios tipos.[13] En la Figura 2.12 se pueden apreciar imágenes microscópicas de dichos mecanismos.

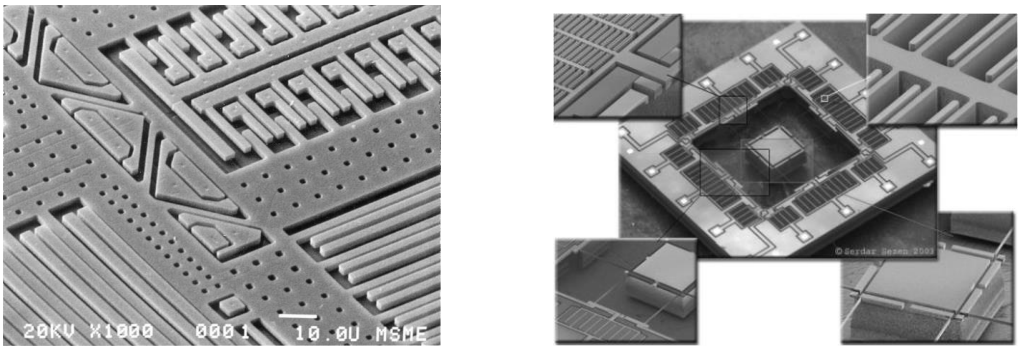


Fig. 2.12. Imágenes microscópicas de los mecanismos de sensado un giroscopio.[40]

2.3. Giroscopio digital ITG-3200A [14]

1. Descripción

El ITG-3200A consiste de tres giroscopios MEMS vibratorios independientes, los cuales detectan velocidad angular alrededor de los ejes *X* (ϕ: alabeo, inglés: *roll*), *Y* (θ: inclinación, inglés: *pitch*) y *Z* (ψ: deriva, inglés: *yaw*), siendo *X* el vector que apunta hacia el frente del dispositivo móvil, *Z* el vector ortogonal al horizonte, con dirección positiva apuntando desde el azimut y *Y* el vector ortogonal a los dos anteriores, como se puede observar en la Figura 2.14:

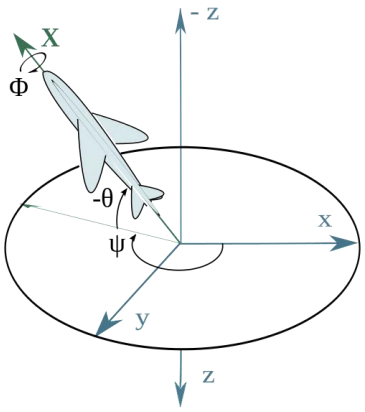


Fig. 2.14: Vectores de navegación de un sistema móvil.

Cuando los giroscopios son rotados alrededor de cualquiera de estos ejes, el efecto de Coriolis causa una deflexión la cual es detectada por un cambio capacitivo. La señal resultante es amplificada, demodulada y filtrada para producir un voltaje que es proporcional a la velocidad angular alrededor de un eje dado. Este voltaje es posteriormente digitalizado utilizando convertidores analógico-digitales de 16 bits embebidos para muestrear cada eje. El rango completo del giroscopio está preseleccionado a 2000 grados por segundo (deg/s). La frecuencia de muestreo del ADC puede programarse por el usuario desde 3.9 hasta 8000 muestras por segundo, y el filtro pasa bajas seleccionable por el usuario permite cortar un alto rango de frecuencias de ruido.

El ITG-3200A posee tres convertidores analógico-digitales (ADCs) de 16 bits de resolución para digitalizar los voltajes de salida de los giroscopios, un ancho de banda del filtro pasa-bajas seleccionable por el usuario, y una interfaz del protocolo I^2C en modo rápido (400 kHz). Adicionalmente posee un sensor de temperatura embebido y un oscilador interno con hasta 2% de error.

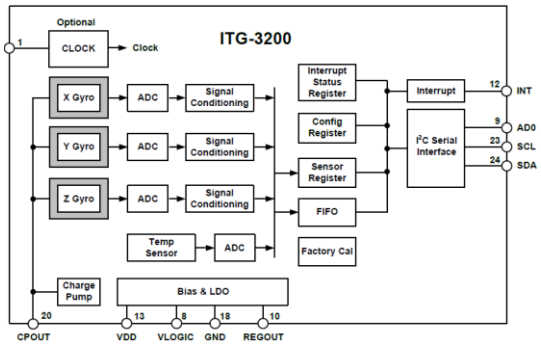


Fig. 2.13: Diagrama de bloques del ITG-3200. [14]

2. Características eléctricas y físicas.

A continuación se presentan los principales parámatros eléctricos del giroscopio seleccionado. Nótese el bajo consumo de corriente y el amplio rango de velocidades muestreables.

Parámetro	Típico	Limite	Unidades
Voltaje de alimentación	2.5	2.1 a 3.6	V
Corriente de alimentación	6.5	-	mA
Corriente en modo de bajo consumo	5	-	μA
Temperatura operacional	-	-40 a 85	°C
Salida en gravedad cero	±40	-	deg/s
Rango completo	±2000	-	deg/s
Sensibilidad	14.375	-	lsb/deg/s
Velocidad de muestreo	8	-	kHz

Tabla 2.2: Características operativas del ITG-3200A. [14]

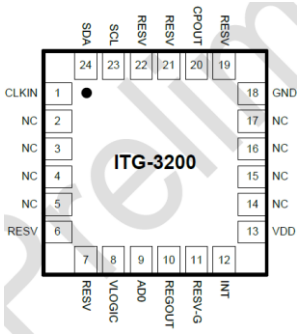


Fig. 2.15: *Pinout* del ITG-3200A.

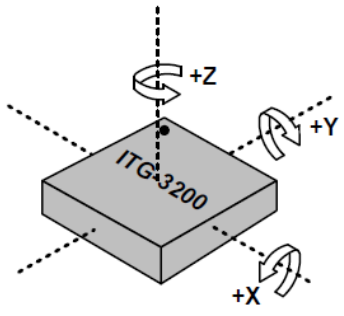


Fig. 2.16: Orientación axial del ITG-3200A.

3. Interfaz de comunicación serial.

El ITG-3200A se comunica a un procesador o sistema maestro mediante la interfaz serial I^2C, en la que el dispositivo siempre actúa como esclavo al comunicarse con el procesador del sistema. La dirección de esclavo (AD) del ITG-3200A es b110100X, la cual es de 7 bits de largo. El LSB (Bit Menos Significativo, inglés: *Less Significat Bit*) de la dirección de 7 bits es determinado por el nivel lógico del pin 9 del encapsulado. Ya que el ITG-3200A siempre opera como esclavo al momento de transmitir, el procesador debe actuar siempre como maestro. Las líneas SDA y SCL suelen requerir resistores *pull-up* a VDD. La máxima velocidad de transmisión con este protocolo es de 400 kHz.

Para escribir a los registros internos del ITG-3200A, el maestro debe transmitir una condición de inicio (S), seguida por la dirección I^2C del esclavo y el bit de escritura (0). En el noveno ciclo de oscilación (en flanco alto), el ITG-3200A reconoce la transferencia, entonces el maestro escribe la dirección del registro (RA) en el bus. Después de que el dispositivo reconoce la recepción de la dirección del registro, el maestro pone los datos a escribir en el bus, lo cual es recibido nuevamente con una señal de reconocimiento (ACK, del inglés: *acknowledge*) y la transmisión de datos puede darse por terminada utilizando la condición de paro (P). El proceso puede ser visualizado en la Figura 2.17

Maestro	S	AD+W		RA		**DATA**		P	
Esclavo			ACK		ACK		ACK		

Fig. 2.17: Secuencia de escritura a un registro mediante el protocolo I^2C con el ITG-3200.

Por otro lado, si se requiere leer cualquiera de los buffers donde se almacenan los datos periódicamente dentro del mismo ITG-3200A, o bien los registros de configuración, el maestro debe comenzar con la condición de inicio (S), seguido de la dirección del giroscopio y un bit de escritura (0). En el noveno ciclo, el giroscopio reconoce la instrucción y manda la señal al maestro. Entonces, el maestro escribe en el bus la dirección del registro que desea leer, a lo cual de nuevo el esclavo manda la señal de reconocimiento junto con los datos solicitados. La comunicación termina cuando el maestro envía un bit de no reconocimiento (NACK) y realiza la condición de paro (P). El proceso puede visualizarse en la Figura 2.18:

Maestro	S	AD+W		RA			NACK	P	
Esclavo			ACK		ACK	**DATA**			

Fig. 2.18: Secuencia de lectura de un registro o buffer mediante el protocolo I^2C con el ITG-3200.

El protocolo I^2C será discutido detalladamente en la Sección 4 del Capítulo 3.

3. SILICON RETINA

3.1. Introducción

La retina es una parte del sistema nervioso central de los animales vertebrados la cual juega un papel importante en las etapas tempranas de procesamiento de información visual. Las imágenes proyectadas en un arreglo fotorreceptor bidimensional (2D) son transducidas a señales eléctricas, e información importante requerida por el cerebro para expresar funciones visuales de orden superior son extraídas por el circuito retinal. Las operaciones realizadas por la retina son mucho más complicadas que aquellas realizadas por un sensor visual CMOS, como se sugiere por la variedad de neuronas sensibles a variaciones de luz. Sin embargo, se pueden encontrar dos tipos de respuestas fundamentales, una es de tipo sostenido o de estado permanente, donde las células mandan impulsos eléctricos continuamente mientras las condiciones de iluminación permanecen constantes, el otro tipo son las de estado transitorio, en las que las células mandan un impulso eléctrico cuando las condiciones de iluminación cambian. Se cree que la respuesta en estado permanente es importante para percibir los objetos estáticos y la respuesta en estado permanente para detectar los objetos en movimiento (Figura 2.19).

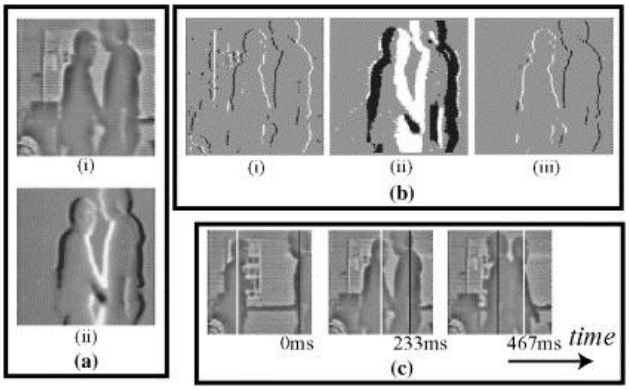


Fig. 2.19: Respuestas en estado transitorio y permanente de la *Silicon Retina*.[15]

Inspirados por la arquitectura única y el algoritmo del circuito de la retina [15] (Figura 2.20), las retinas de silicón neuromórfico, las cuales son novedosos arreglos analógicos dentro de los circuitos de gran escala (*VLSI circuits*, inglés: V*ery Large Scale Integrated circuits*) han sido fabricados. Algunas retinas de silicón emulan el campo receptivo del campo exterior de la retina; por otro lado, han sido fabricadas retinas de silicón imitando la respuesta transitoria para detectar objetos en movimiento. Sin embargo, la mayoría de los chips fabricados hasta la fecha emulan solamente la respuesta permanente o la respuesta transitoria exclusivamente, lo cual limita las aplicaciones de la retina de silicón en situaciones donde se requiere procesamiento de imagen en tiempo real. La *Silicon Retina* provee datos analógicos imitando a ambas la respuesta transitoria y la permanente.

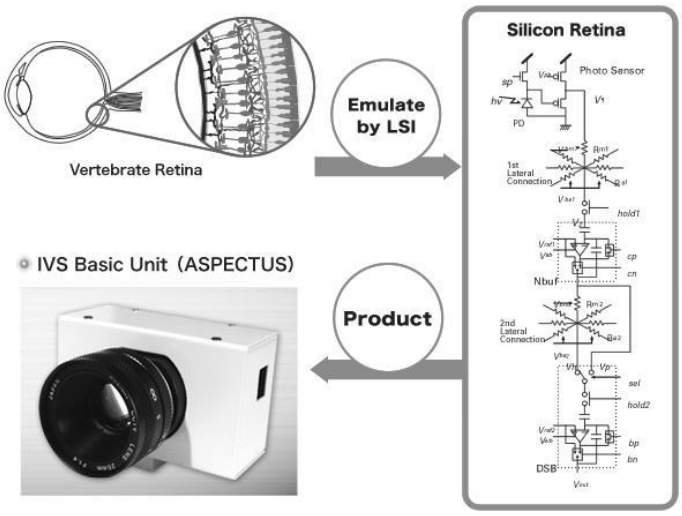


Fig. 2.20: El principio de operación de la *Silicon Retina* está inspirado en la retina humana. [15]

3.2. Descripción del circuito

La salida del chip emulando la respuesta en estado permanente posee un campo receptivo tipo Gaussiano-Laplaciano (∇^2G) y, por lo tanto, lleva a cabo un alisado y aumento de contraste en las imágenes de entrada. La salida del chip emulando la respuesta en estado transitorio se obtiene sustrayendo imágenes consecutivas que son alisadas en un principio por una red resistiva.

Existen dos tipos fundamentales de procesamiento de imagen que pueden lograrse mediante este chip controlando las señales externas mediante diferentes tiempos. La *Silicon Retina* es controlada mediante un FPGA utilizando convertidores ADC y DAC para intercambiar información con los buffers y los amplificadores de las redes resistivas (Figura 2.22). Una breve descripción de las ventajas de los FPGA será dada en la próxima sección.

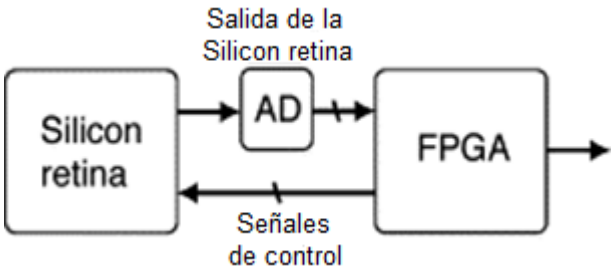


Fig. 2.22: Intercambio de información entre FPGA y Silicon Retina

La imagen de la Figura 2.21 muestra el diseño del circuito de un solo pixel. Este circuito fue diseñado basándose en el modelo de la retina de los animales vertebrados después de detalladas observaciones fisiológicas. El chip consiste de dos capas de redes resistivas, las cuales poseen diferentes valores para las resistencias de acoplamiento entre los pixeles colindantes. Una sustracción de estas dos redes resistivas

genera el campo receptivo tipo $\nabla^2 G$. El sensor fotorresistivo es un sensor activo de pixel (APS, inglés: *Active Pixel Sensor*) el cual consiste de un fotodiodo y un amplificador operacional en configuración de seguidor (Véase Figura 2.21). El APS tiene una alta sensibilidad porque acumula fotoelectrones en el capacitor parasítico del fotodiodo. Además, el rango dinámico puede ser controlado cambiando la capacitancia en el tiempo.

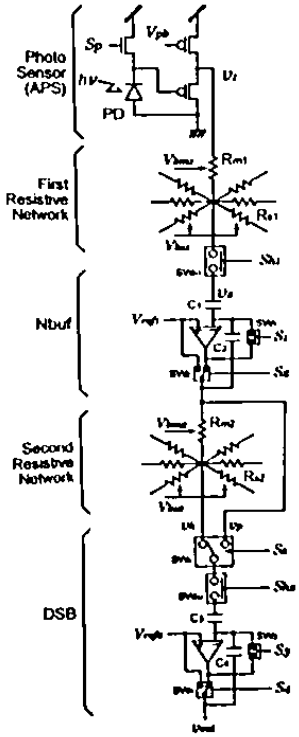


Fig. 2.21: Red resistiva de un solo pixel en la *Silicon Retina*. [15]

Los pixeles colindantes son conectados mediante resistores MOS en las redes resistivas primera y segunda. Las resistencias de los resistores MOS son controlables mediante voltajes externos. Los circuitos nombrados NBUF (Buffer N) y DSB (Transmisores de Doble Banda, inglés: *Double SideBand Transmitters*) son circuitos de tipo *track and hold*.

El chip cuenta con una función de ahorro de energía para reducir el consumo de potencia. Cuando el modo está activo, no se provee de corriente a los Nbufs ni a los DSBs, excepto a los DSBs en una columna a la vez, la cual es seleccionada por un registro de desplazamiento cada vez que se lee la matriz de pixeles. Cuando el modo está desactivado se provee de corriente a todos los Nbufs y DSBs todo el tiempo. La versión utilizada en este proyecto del chip tiene 128x128 pixeles. El chip fue implementado con tecnología CMOS estándar analógica, de doble polímero y triple metal de 0.6 um; el tamaño del dado fue de 8.9 x 8.9 mm^2. El consumo de energía sin el modo de ahorro es de 267.9 mW con una fuente de voltaje de 3.3 V; con la función encendida este valor desciende a solamente 20.0 mW. Las especificaciones completas del chip se pueden apreciar en la Tabla 2.3.

Proceso	CMOS 0.8 um de doble polímero y triple metal
Tamaño del dado	8.9 x 8.9 [mm^2]
Número de pixeles	127(H) x 127(V) [pixel]
Área de pixeles (área PD)	178.650 x 154.725 [um^2] 2:√3 (37.575 x 23.100 [um^2])
Factor de llenado	3.14 [%]
Consumo de energía (sin modo de ahorro) (con modo de ahorro)	 367.9 mW (200.0 uW / pixel) @ 3.3 V 40.0 mW (21.7 uW / pixel) @ 3.3 V

Tabla 2.3: Especificaciones de la *Silicon Retina.* [15]

4. CONTROLADOR FPGA

4.1. Hardware

Un FPGA (Arreglo de Compuertas Programables en Campo, inglés: *Field Programmable Gate Array*) es un dispositivo semiconductor que puede ser programado después de su manufactura. En vez de estar restringido a una función específica de hardware, un FPGA le permite al usuario programar funciones y características, adaptarse a nuevos estándares y reconfigurar el hardware para aplicaciones especificas aun después de que el producto ha sido instalado en su aplicación final, de ahí el nombre "Programable en Campo". Se puede utilizar un FPGA para implementar cualquier función lógica que un ASIC pueda desempeñar (Circuito Integrado de Aplicación Específica, inglés: *Application Specific Integrated Circuit)*, pero el contar la capacidad de mejorar la funcionalidad después de haber instalado el producto ofrece una ventaja definitiva en muchas aplicaciones.

A diferencia de los FPGA de generaciones anteriores, los cuales solo utilizaban pines de entrada/salida con interconexiones y programación de compuertas lógicas, los FPGA actuales consisten de una mezcla de diferentes periféricos, tales como:

- Memoria SRAM embebida
- Transceptores de alta velocidad
- Pines de entrada/salida de alta velocidad
- Bloques lógicos
- Bloques multimedia

En específico, un FPGA contiene componentes de programación lógica, llamados LE (Elementos Lógicos, inglés: *Logic Elements*) y una jerarquía de conexiones reconfigurables que permiten a los LEs conectarse físicamente. Los LEs pueden ser configurados para desempeñar funciones combinacionales complejas, o como simples compuertas AND y XOR [16]. En la mayoría de los FPGAs, los bloques lógicos también incluyen elementos de memoria, los cuales pueden ser desde *flip-flops* hasta memorias de almacenamiento masivo (Figura 2.23).

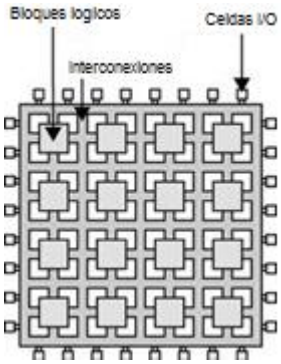


Fig. 2.23: Arquitectura de un FPGA en un nivel básico. [16]

Xilinx y Altera son actualmente los líderes de mercado de FPGAs y a la vez los más grandes rivales [17]. Juntos controlan más del 80% del mercado total. Ambos fabricantes proveen plataformas de diseño gratuitas para los sitemas operativos Windows y Linux para sus dispositivos.

Hasta 2010, Xilinx ofrecía dos familias principales de FPGA [17]: la familia de alto desempeño, *Virtex* y la familia de alto volumen *Spartan*. Cada modelo ha sido lanzado en diferentes versiones desde sus inicios. Con la introducción de la tecnología de FPGAs de 28 nm en Junio 2010, Xilinx reemplazo la familia Spartan con dos nuevas familias: La *Kintex* y la *Artix*. Estas nuevas familias incorporan mejoras en la potencia del sistema, desempeño, capacidad y precio. Estas nuevas familias proveen un 50 por ciento de reducción en el consumo de energía comparados con los dispositivos de 40 nm y ofrecen hasta 2 millones de celdas lógicas por dispositivo.

La familia de dispositivos *Spartan-3* se basa en el éxito obtenido de la anterior familia *Spartan-IIE* incrementando la cantidad de recursos lógicos, la capacidad de la memoria RAM interna, el número total de pines de entrada/salida y el nivel de desempeño en general así como las funciones de manejo del oscilador principal. Varias mejoras derivan del desarrollo de la tecnología de la familia de dispositivos *Virtex-II* .

Estas mejoras aplicadas a la familia de FPGA *Spartan-3,* combinados con una tecnología de procesamiento avanzada, entregan una funcionalidad y ancho de banda por dólar mucho mas alta de lo que era previamente posible, estableciendo nuevos estándares en la industria de la lógica programable. Debido a su costo excepcionalmente bajo, los FPGA *Spartan-3* son aptos para un amplio rango de aplicaciones de electrónica de consumo; incluyendo acceso a internet de banda ancha, sistemas domóticos, monitores/proyectores y equipo de televisores digitales.

4.2. Lenguajes de programación [18]

Actualmente, existen dos lenguajes descriptivos estandarizados en la industria, VHDL y Verilog. La creciente complejidad de los diseños de FPGA ha conllevado un incremento en el número de consultores especializados con herramientas específicas y con librerías y macros propios escritos ya sea en VHDL o Verilog. Como resultado de esto, es importante que el diseñador de FPGA conozca ambos lenguajes y que los vendedores de dichas herramientas y librerías provean un entorno de programación el cual sea capaz de recibir indistintamente ambos lenguajes. La misma funcionalidad se puede lograr con ambos lenguajes, tanto en estructura de hardware como en modelos abstractos. La elección de qué lenguaje utilizar, por lo tanto, no reside tanto en la capacidad técnica del mismo sino en:

- Preferencia personal.
- Disponibilidad de herramientas de los desarrolladores.
- Cuestiones comerciales y de mercadeo.

El modelado de estructuras con VHDL y Verilog cubren diferentes espectros a través de las jerarquías de los FPGAs, véase la Figura 2.24:

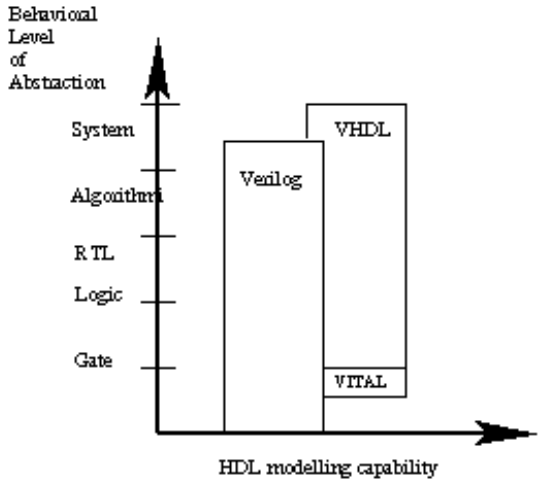


Fig. 2.24. Capacidades de modelado de los lenguajes de programación de FPGA [18]

A continuación compararemos ambos lenguajes en 3 categorías fundamentales: Compilación, estructuras de datos y reusabilidad de los códigos.

1. Compilación

VHDL: Múltiples unidades de diseño (pares de entidad/arquitectura) que residen en la misma carpeta del sistema pueden compilarse por separado si así se desea. Sin embargo, se acostumbra dejar a cada unidad de diseño en su propio archivo del sistema en cuyo caso la compilación sea junta o separada no presenta ningún problema.

Verilog: El lenguaje Verilog tiene sus raíces en el modo interpretativo similar al de un lenguaje como C. Por lo tanto, se debe tener cuidado en el orden del código a la hora de compilar, especialmente si se tienen proyectos con múltiples unidades de programación. El resultado final de la simulación puede variar si se cambia el orden de la compilación.

2. Tipos de datos

VHDL: Una multitud de tipos de datos definidos por el usuario pueden ser utilizados. Esto puede significar también declarar funciones dedicadas de conversión entre distintos tipos de datos. La elección de qué tipo de dato utilizar se debe considerar detenidamente, especialmente con tipos de datos abstractos; esto hará que los modelos sean más fáciles de escribir y leer, además de evitar conversiones innecesarias entre tipos. VHDL puede ser preferido por varios diseñadores precisamente por la libertad en la declaración de tipos de datos disponible.

Verilog: Comparado con VHDL, los tipos de datos de Verilog son extremadamente sencillos, fáciles de utilizar y diseñados específicamente para modelar estructuras de hardware más que estructuras abstractas. A diferencia de VHDL, todos los tipos de datos están definidos dentro de las librerías de Verilog y no pueden definirse por el usuario. Cada tipo de dato tiene un correspondiente en el circuito eléctrico. Verilog puede ser preferido por algunos diseñadores por su simplicidad.

3. Reusabilidad de los códigos

VHDL: Los procedimientos y las funciones pueden ponerse en un paquete de modo que se hagan disponibles para cualquier unidad de diseño que quiera utilizarlas.

Verilog: No existe el concepto de paquetes en Verilog. Las funciones y procedimientos utilizados en un modelo deben ser definidos en el mismo. Para hacer funciones y procedimientos disponibles para diferentes módulos, los mismos deben ponerse en una carpeta de sistema específica y ser incluidos al inicio del código, de manera muy similar a como se incluyen librerías y otros códigos en C.

Empezando sin ningún conocimiento previo de ninguno de los lenguajes de programación, Verilog es probablemente el más sencillo de comprender y utilizar. VHDL puede parecer menos intuitivo al inicio por dos razones principales. Primero, es un lenguaje fuertemente estructurado, una característica que lo hace robusto y potente para los usuarios avanzados después de una larga curva de aprendizaje. Segunda, hay muchas diferentes maneras de modelar el mismo circuito, especialmente aquellos con estructuras jerárquicas de tamaño considerable.

5. CONTROLADOR *PSOC 1*

5.1. Introducción

PSoC ó Sistema de Programación en Chip (inglés: *Program System on Chip*) es una nueva tecnología aplicada al desarrollo de los microcontroladores, la cual permite al diseñador seleccionar bloques analógicos o digitales de diferentes dispositivos electrónicos , ya sean estos analógicos y/o digitales para luego programarlos mediante un lenguaje de programación híbrido el cual combina programación gráfica mediante bloques y lenguaje C o ensamblador [19]. Este tipo de tecnología resulta muy versátil ya que permite reconfigurar los pines de entrada/salida del microcontrolador lo cual nos brinda la posibilidad de reducir la cantidad de componentes externos necesarios para el procesamiento de señales analógicas o la implementación de contadores y relojes externos, por ejemplo. En la Figura 2.25 debajo se observa el diagrama de bloques de la organización de un chip PSoC:

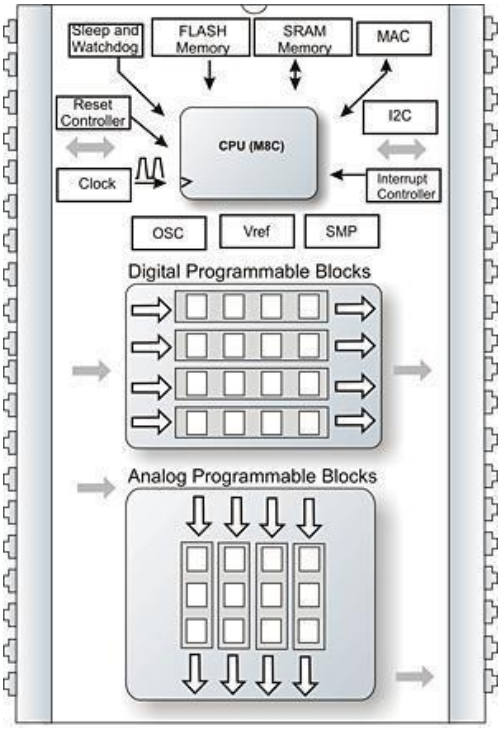


Fig. 2.25: Estructura de bloques programables de un chip PSoC.[19]

Las características principales de los microcontroladores PSoC son [19]:

- Unidad multiplicadora MAC.
- Multiplicación por hardware de 8x8 con almacenamiento de 32 bits.
- Operación con varios voltajes de alimentación: 5, 3.3 o hasta 1 V_{DC} en modo de ahorro de energía.
- Oscilador interno de frecuencia seleccionable o externo .
- Voltaje de referencia variable para adaptarse a distintos sensores.

5.2. Sistema de bloques

La programación mediante bloques es una de las características principales que diferencian a la plataforma PSoC de otras similares. Un bloque se define como la unidad programable más básica que puede funcionar independientemente asumiendo que se le provee de una señal de reloj y otras entradas. La cantidad de bloques de la que dispone un solo chip depende de la familia a la que este pertenece. Las funciones que un solo bloque puede realizar son, entre otras [19]:

- 16 kbytes de memoria programable.
- 256 Mb de RAM.
- Conversores ADC con resolución de hasta 14 bits.
- Conversores DAC con resolución de hasta 9 bits.
- Amplificadores de ganancia variable.
- Amplificadores inversores.
- Filtros Analógicos.
- Timers de 8, 16 y 32 bits.
- Moduladores de ancho de pulso (PWM) DE 8, 16 ó 32 bits.
- Interfaces de comunicación UART, SPI, I^2C.

Los bloques de PSoC pueden ser divididos en dos grandes grupos de acuerdo a su función, analógicos y digitales:

1. **Bloques Analógicos:** Los bloques analógicos están agrupados en columnas, la cantidad de éstas depende de la familia del microprocesador utilizado, pudiendo ser 3, 4 ó 5 y cada columna posee 3 bloques. Cada bloque posee una entrada, una referencia y una salida. Estos bloques poseen la ventaja de que se puede acceder directamente a ellos desde el exterior. En la Figura 2.26 debajo observamos los multiplexores de entrada y el buffer de salida, dispositivos mediante los cuales podemos referenciar y enviar las señales procesadas a cualquier pin de salida o a un bloque digital para otros propósitos de procesamiento. Nótese que la capacidad programable de cada columna está limitada a tres bloques, los cuales comparten el oscilador y la referencia.

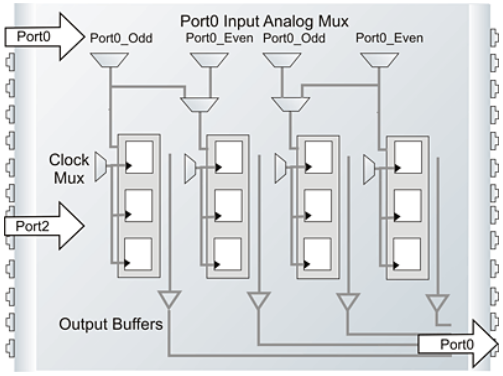


Fig. 2.26: Bloques analógicos del PSoC 1.[19]

2. **Bloques Digitales:** Dentro de los bloques digitales, existen dos tipos de bloques: DCB (la C indica que es para componentes de comunicación como UART, SPI, etc.) y DBB, los cuales son de tipo general. De igual forma que los bloques analógicos, la cantidad de bloques disponibles depende de la familia del microcontrolador seleccionada.

A diferencia de los bloques analógicos, estos no pueden ser accesados directamente desde el exterior, la señal que llega a un bloque digital tuvo que haber pasado antes por un bloque analógico, si no se requiere ningún tipo de acondicionamiento de señales se puede programar en configuración de seguidor. Los bloques digitales pueden desempeñar tareas de comunicación (UART, SPI, I2C, etc.) o de procesamiento digital (ADC, DAC, PWM, etc.). Algunos recursos como los conversores ADC consumen ambos tipos de bloques.

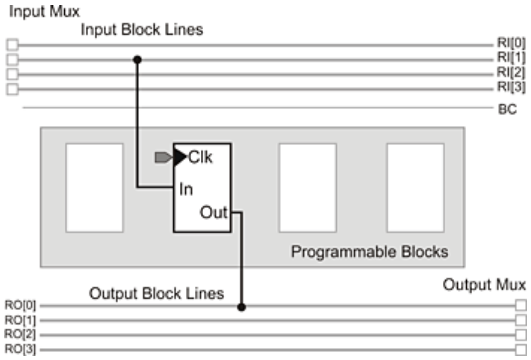


Fig. 2.27: Un bloque digital del PSoC 1

Al seleccionar un componente digital, como se muestra en la Figura 2.27, dependiendo de sus características (esto es, el número de contadores y otros recursos internos que el mismo requiere para desempeñar su función) puede ocupar 1, 2 o 3 bloques, los cuales pueden asentarse en cualquier bloque disponible y éstos se denominan por las siglas DB ó DC más el número de su ubicación. La nomenclatura se ocupa después cuando se requiere programar interrupciones y otros comandos mediante software**.**

La salida de los bloques digitales se envía mediante un sistema de hilos a la salida correspondiente. En la Figura 2.28 debajo se muestran las líneas globales para la comunicación de sistemas digitales entre las líneas de entrada y los multiplexores. Éstas se encuentran divididas en 2 grupos los cuales se separan en las líneas pares (GIO) tal como P1(2) y líneas impares (GIE), como por ejemplo, P1(3) (estos corresponden al pin 2 del puerto 1 y el pin 3 del puerto 1, respectivamente).

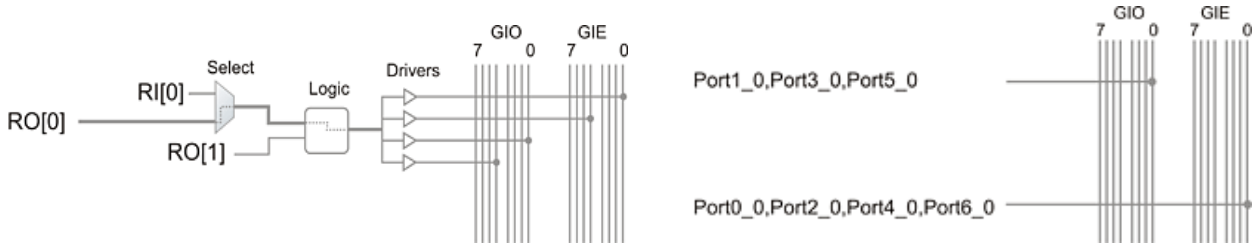


Fig. 2.28: Multiplexores para programar las entradas y salidas

5.3. Programación híbrida

La interfaz de programación para la plataforma PSoC es un programa llamado *PSoC Designer,* el cual es distribuido por *Cypress Semiconductor* de manera gratuita. Esta interfaz contiene tres principales maneras de programar los diferentes recursos que posee un chip PSoC: Programación de bloques, programación de código y configuración de parámetros.

1. **Programación de bloques:** La interfaz gráfica de *PSoC Designer* permite seleccionar a los bloques analógicos o digitales a utilizar de acuerdo a su función, asociar el oscilador y otras señales necesarias e interconectar diferentes bloques a voluntad del diseñador y muestra un indicador de recursos consumidos como se aprecia en la Figura 2.29:

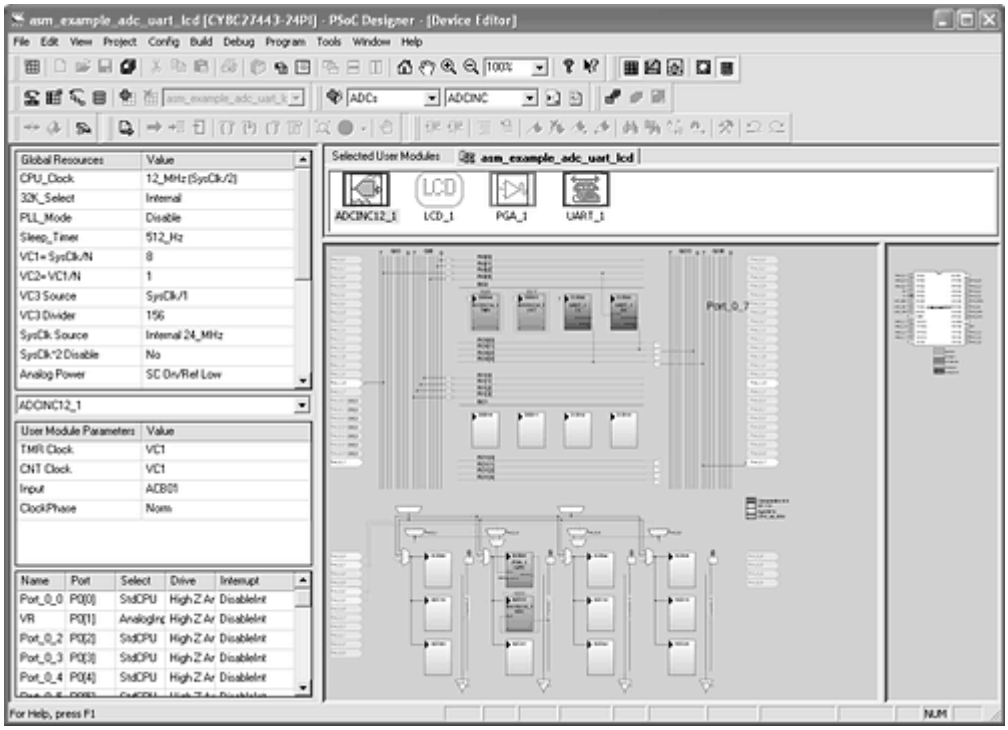


Fig. 2.29: Programación gráfica mediante bloques.

Es importante destacar que al seleccionar un bloque, de inmediato el programa muestra un archivo PDF con la configuración e instrucciones básicas para programar dicho bloque. Esto facilita el auto aprendizaje y, por lo tanto, expande la base de usuarios de la plataforma.

Cada vez que se selecciona algún componente se consume una cierta cantidad de memoria RAM del microcontrolador, la cual es mostrada en el indicador de recursos consumidos, el cual se aprecia en la Figura 2.30:

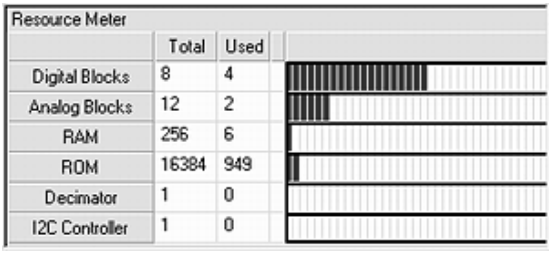


Fig. 2.30: Indicador de recursos consumidos

2. **Programación de Código:** La interfaz de programación provee de un editor de texto donde se ingresa el código a ejecutar en lenguaje ensamblador o C (ejecutado mediante un compilador), como se muestra en la Figura 2.31. La plataforma posee una serie de librerías y comandos preprogramados los cuales son relativamente sencillos de utilizar y se puede aprender a aplicarlos leyendo los archivos anexos a cada bloque que se discutieron previamente.

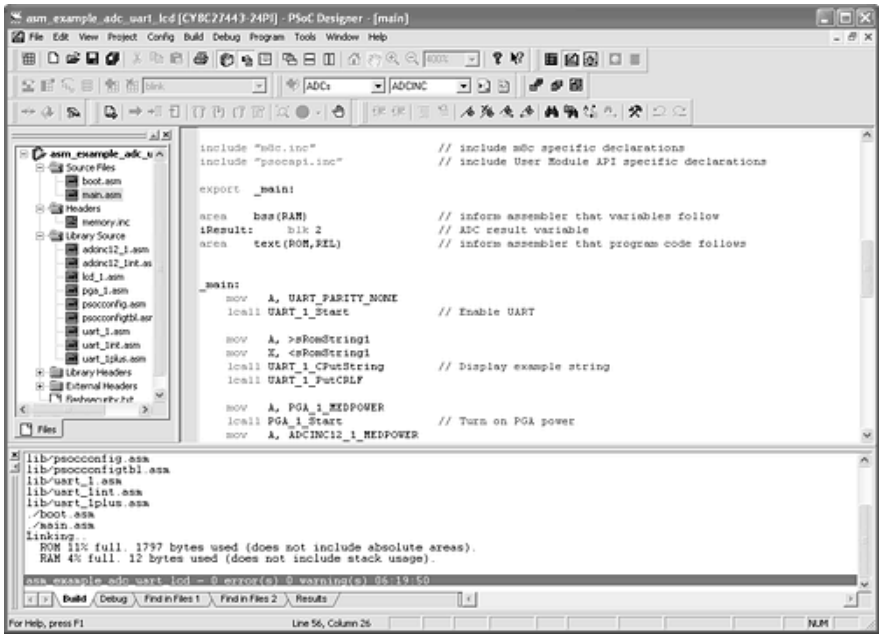
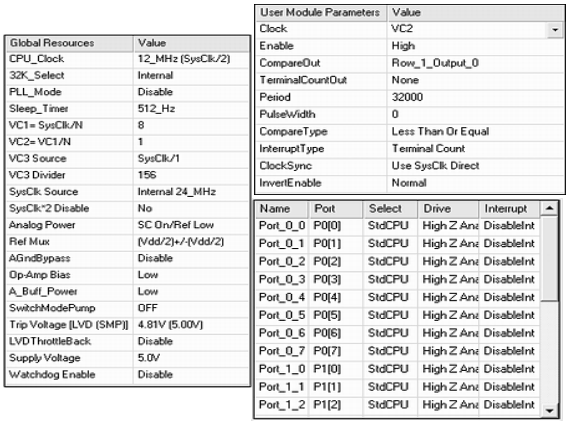


Fig. 2.31: Interfaz de programación en lenguaje C

3. **Configuración de parámetros:** Finalmente, algunos parámetros de los bloques tanto analógicos como digitales son configurables mediante una serie de ventanas con diferentes opciones para seleccionar.

Por un lado, se deben seleccionar las configuraciones globales del microcontrolador, tales como la frecuencia u origen del oscilador, el tipo de interrupciones disponibles, el voltaje de referencia, entre otros. Además, cada bloque, ya sea analógico o digital también posee características individuales que se programan mediante estas ventanas de configuración (por ejemplo, si escogimos un *timer* se debe configurar su velocidad de conteo, interrupciones, modo de operación, etc.)

Finalmente, se deben de configurar los pines de entrada y salida del microcontrolador para tener un desempeño óptimo del sistema, funciones tales como resistencias en *pull-up* o entradas de alta impedancia se seleccionan en estas ventanas. Todos estos parámetros configurables se pueden observar en la Figura 2.32 abajo:



Global Resources	Value
CPU_Clock	12_MHz (SysClk/2)
32K_Select	Internal
PLL_Mode	Disable
Sleep_Timer	512_Hz
VC1= SysClk/N	8
VC2= VC1/N	1
VC3 Source	SysClk/1
VC3 Divider	156
SysClk Source	Internal 24_MHz
SysClk*2 Disable	No
Analog Power	SC On/Ref Low
Ref Mux	(Vdd/2)+/-(Vdd/2)
AGndBypass	Disable
Op-Amp Bias	Low
A_Buff_Power	Low
SwitchModePump	OFF
Trip Voltage [LVD (SMP)]	4.81V (5.00V)
LVDThrottleBack	Disable
Supply Voltage	5.0V
Watchdog Enable	Disable

User Module Parameters	Value
Clock	VC2
Enable	High
CompareOut	Row_1_Output_0
TerminalCountOut	None
Period	32000
PulseWidth	0
CompareType	Less Than Or Equal
InterruptType	Terminal Count
ClockSync	Use SysClk Direct
InvertEnable	Normal

Name	Port	Select	Drive	Interrupt
Port_0_0	P0[0]	StdCPU	High Z Ana	DisableInt
Port_0_1	P0[1]	StdCPU	High Z Ana	DisableInt
Port_0_2	P0[2]	StdCPU	High Z Ana	DisableInt
Port_0_3	P0[3]	StdCPU	High Z Ana	DisableInt
Port_0_4	P0[4]	StdCPU	High Z Ana	DisableInt
Port_0_5	P0[5]	StdCPU	High Z Ana	DisableInt
Port_0_6	P0[6]	StdCPU	High Z Ana	DisableInt
Port_0_7	P0[7]	StdCPU	High Z Ana	DisableInt
Port_1_0	P1[0]	StdCPU	High Z Ana	DisableInt
Port_1_1	P1[1]	StdCPU	High Z Ana	DisableInt
Port_1_2	P1[2]	StdCPU	High Z Ana	DisableInt

Fig. 2.32: Diferentes parámetros configurables en la interfaz de programación.

5.4. Chip CY8C29466

La plataforma utilizada en este proyecto en particular es la CY8C29466 [20], de la familia PSoC 1. La familia PSoC 1 es la más sencilla de las tres y se caracteriza por tener chips sencillos de topografía Harvard con palabras de 8 bits y recursos de RAM limitados a cambio de un precio más accesible. Las familias superiores ofrecen una mayor cantidad de pines, recursos y topografías más complejas. En la Figura 2.33 debajo se aprecia un esquema donde se relacionan el desempeño y la topografía de las diferentes familias.

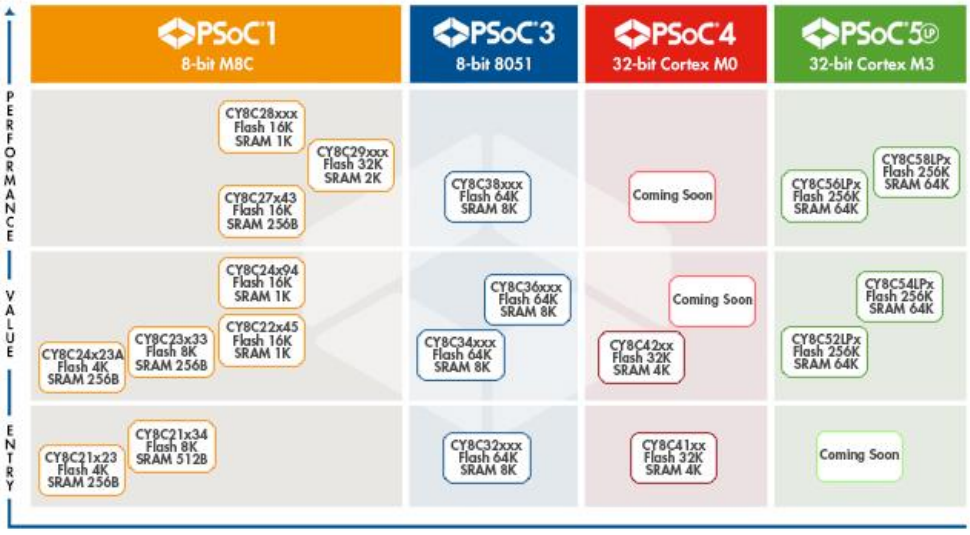


Fig. 2.33: Familias actuales de PSoC de *Cypress Semiconductor*.[41]

5.5. Funciones

A continuación se explicará a detalle las funciones clave del PSoC utilizadas en este proyecto.

1. ADC en el PSoC™ 1 [21]

El paquete ADCINC es un ADC pseudo-diferencial o de un solo polo que entrega un numero discreto de entre 6 y 14 bits. La máxima frecuencia de oscilación que puede recibir es de 8MHz, pero 2MHz es la mayor frecuencia recomendada si se requiere linealidad en los datos. Este ADC solo puede ser utilizado una vez por microcontrolador, debido a su implementación la cual utiliza el decimador de hardware en vez de un bloque digital. Este es el ADC más eficiente en cuanto al uso de los recursos del sistema. Un modulador de segundo orden puede ser implementado agregando un bloque de capacitor/interruptor, permitiendo una mejor linealidad al utilizar una frecuencia de 8MHz.

Los tiempos de muestreo son implementados mediante PWM (Inglés: *Pulse Width Modulation*, español: Modulación de Ancho de Pulso) de ocho bits el cual es síncrono con el muestreo de la entrada. El bloque ADCINC requiere 2n-1 ciclos de integración para generar una salida de n bits de resolución.

El bloque ADCINC provee de un modulador de primer orden formado por un bloque PSoC de un switch con capacitor analógico, un bloque PSoC digital y el decimador, como se muestra en la Figura 2.34:

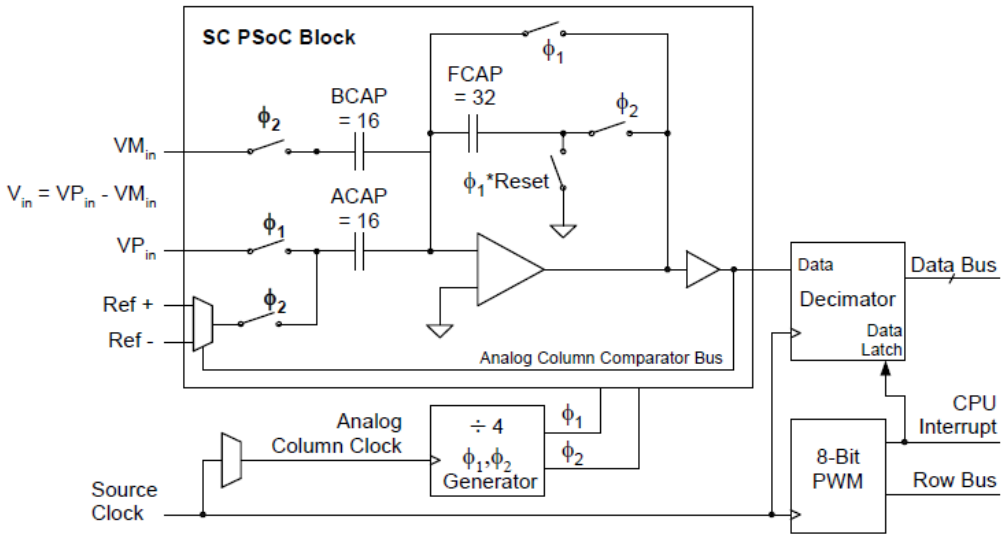


Fig. 2.34: Figura esquemática del bloque ADCINC con modulador de primer orden.[21]

El rango del bloque ADCINC esta dado en un rango de $\pm V_{Ref}$, donde V_{Ref} es determinado por el diseñador en la ventana de Recursos Globales del programa *PSoC Designer*. Para una escala fija, V_{Ref} esta referido a $V_{Bandgap}$ o $1.6*V_{Bandgap}$. Para escala variable esta referido a Port2[6].

La Tabla 2.4 debajo muestra las características de DC y AC para el modulo ADCINC del PSoC1:

Parámetro	Típico	Limite	Unidades
Rango de voltaje de entrada	-	Vss a Vdd	V
Capacitancia de entrada	3	-	pF
Impedancia de entrada	1/(C*clk)	-	Ω
Resolución	-	8 a 14	Bits
Velocidad de muestreo	-	0.125 a 31.25	Ksps
SNR	44	-	dB
DNL	0.6	-	LSB
INL	0.7	-	LSB
Error de offset	10	-	mV
Error de ganancia	3.0	-	%FSR
Corriente operacional	-	50 a 1900	μA
Velocidad del oscilador	-	0.032 a 8	MHz

Tabla 2.4: Características eléctricas de DC y AC del modulador de primer orden de 5.0V.

2. SPI en el PSoC™ 1 [22]

Características generales:

- ✓ Soporta el protocolo maestro SPI.
- ✓ Soporta los modos 0, 1, 2 y 3 definidos en el estándar SPI.
- ✓ Capacidad de elegir la fuente del reloj y el pin de MISO (*Master-In Slave-Out*).
- ✓ Capacidad de elegir los pines de salida para MOSI (*Master-Out Slave-In*) y SCLK (*Slave Clock*).
- ✓ Interrupciones programables del bloque SPI.

El bloque de usuario SPIM es el Maestro de Interconexión Serial de Periféricos(inglés: *Serial Peripheral Interface Master*). Puede realizar transferencias de datos síncronas *full duplex* de 8 bits. La polaridad y la fase del reloj del esclavo (SCLK) así como el LSB (bit menos significativo, inglés: *Less Significant Bit*) pueden ser especificados para acomodar a cualquier tipo de reloj del protocolo SPI. La señal de selección de esclavo, la cual es controlada mediante el software que el usuario programa, se puede configurar para seleccionar a uno o más esclavos a la vez. El bloque SPIM del PSoC tiene cableado interno programable para las entradas y salidas y el control es dependiente de las interrupciones del protocolo.

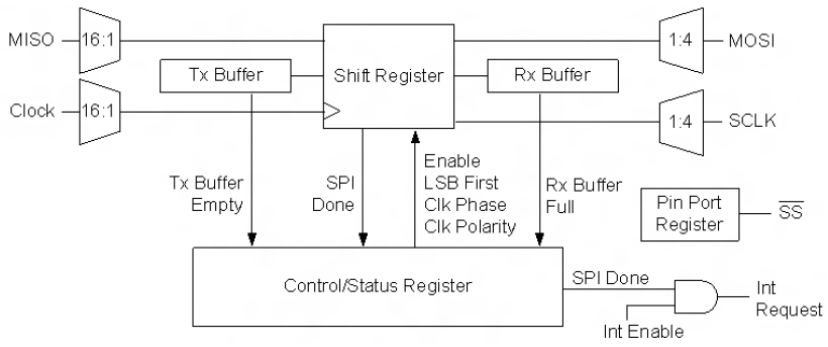


Fig. 2.35: Diagrama del modulo SPIM.[22]

Como se ha dicho anteriormente, SPIM es un modulo de usuario que opera como un maestro en el protocolo SPI, utiliza los registros *Tx Buffer, Rx Buffer, Control* y *Shift* del PSoC y uno o mas registros de los pines de entrada/salida.

El registro de control es inicializado y configurado utilizando el editor de dispositivos y/o las rutinas de la API (interfaz de programación de aplicaciones, inglés: *Application Programming Interface*). La inicialización consta de establecer el modo de operación del reloj para la comunicación con los esclavos, los modos soportados son 0, 1, 2 y 3. Tanto el maestro como el esclavo deben de estar configurados con el mismo reloj para operar correctamente. Los modos SPI se describen en la Tabla 2.5:

Modo	Flanco de SCLK que genera *latch*	Polaridad del oscilador	Notas
0	Alto	Invertido	Flanco alto hace el latch, los datos cambian en flanco bajo
1	Alto	No Invertido	
2	Bajo	Invertido	Flanco bajo hace el latch, los datos cambian en flanco alto
3	Bajo	No Invertido	

Tabla 2.5: Modos de comunicación SPI.

La señal de SCLK es el reloj de transmisión/recepción del protocolo SPI. Esta representado como la mitad de la frecuencia del reloj de entrada (reloj maestro, oscilador externo, etc.), la velocidad de transmisión/recepción efectiva es también la mitad del mismo. El reloj de entrada es definido por el usuario. La señal MISO es la señal que entra al dispositivo maestro desde el esclavo (inglés: *Master-In Slave-Out*), mientras que la señal MOSI es la que sale del maestro hacia el esclavo (inglés: *Master-Out Slave-In*).

El hardware del bloque SPIM transmite datos del dispositivo maestro SPI en el pin definido para MOSI y simultáneamente recibe información en el pin definido para MISO. La misma señal de SCLK es utilizada para ambas operaciones.

3. I^2C en el PSoC™ 1 [23]

El modulo de usuario del protocolo I^2C provee soporte para una configuración de hardware I^2C. El modulo es capaz de transmitir datos a 50/100/400 kbps cuando el CPU está configurado para oscilar a 12 MHz. Velocidades más altas o bajas que esta pueden ser utilizadas, pero pueden resultar en más o menos aglomeración de datos en el bus durante el direccionamiento o la transferencia de datos. La especificación del protocolo I^2C permite al maestro oscilar a una frecuencia máxima de 100 kHz. El modo de direccionamiento de 7 bits es soportado directamente por las librerías del compilador, sin embargo, para direccionamiento de 10 bits es necesario código adicional por parte del usuario. Este módulo no requiere ningún bloque analógico o digital de los recursos del PSoC.

El recurso de I^2C soporta transferencia de datos a un nivel de byte por byte. Al final de cada direccionamiento o transmisión/recepción de datos, el estado de la línea es reportado y una interrupción dedicada puede ser activada. El reporte del estado y la generación de interrupciones depende de la dirección de la transferencia de datos y de la condición del bus detectada por hardware. Las interrupciones pueden ser configuradas para ocurrir en los siguientes casos:

- Cuando se haya recibido un byte completo.
- Cuando se detecte un error en el bus.
- Cuando se pierda la arbitrariación del bus.

El módulo I^2C es capaz de actuar como maestro o como esclavo según sea necesario. Sin embargo, ignorando la separación entre maestros y esclavos, existen dos casos generales, el de un receptor y el de un transmisor.

- Para un receptor en I^2C, una interrupción ocurrirá cuando venga el octavo bit de información. En este punto el dispositivo receptor debe decidir si reconocer (ACK) o no reconocer (NAK) el byte recibido, sea este de dirección o de datos. El receptor debe entonces escribir los bits de control apropiados al registro I2C_SCR, informando acerca del estado de ACK/NAK y recorrer el siguiente byte entrante.
- Para el caso de un transmisor, la interrupción ocurrirá cuando el dispositivo externo haya mandado la señal de ACK/NAK. El registro I2C_SCR debe ser leído para determinar el estado del bit. Para un transmisor, los datos deben ser cargados en el registro I2C_DR y el registro I2C_SCR debe ser reescrito otra vez para disparar la siguiente porción de la transmisión.

Las descripciones detalladas del bus I^2C y de la implementación de los recursos en el PSoC pueden ser obtenidas acudiendo a la especificación completa del protocolo I^2C[32] y a la hoja de datos de PSoC 1.[20]

6. ARDUINO UNO

6.1. Introducción

"Arduino es una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar. Se creó para artistas, diseñadores, aficionados y cualquiera interesado en crear entornos u objetos interactivos."

-- Equipo de Arduino [26]

Arduino es una dispositivo embebido controlado por un microcontrolador diseñado para hacer el proceso de utilizar medios electrónicos en proyectos multidisciplinarios más accesible. El hardware consiste de una tarjeta programadora de código libre diseñada para ser utilizada con un microcontrolador AVR de 8

bits de Atmel. El software consiste de un lenguaje de programación estándar, un compilador y un *boot loader* que ejecuta los comandos programados en el microcontrolador.

Desde su lanzamiento en 2007 han salido varias versiones diferentes de la tarjeta y se estima que ha sido adoptada por más de 300,000 usuarios alrededor del mundo [26]. Al ser un proyecto con licencia libre, muchas compañías pequeñas han lanzado prototipos basados en la misma plataforma con alguna mejora en diseño, precio, funcionalidad, etc.

Arduino está basado sobre la plataforma de código libre *Wiring*, y es programada utilizando un lenguaje similar a C++ con ciertas simplificaciones y modificaciones en las librerías y en la sintaxis. El IDE (entorno de desarrollo integrado, inglés: *Integrated Development Environment*) está basado en otro proyecto de código libre llamado *Processing*.

6.2. Características

Una tarjeta Arduino consiste de un microcontrolador AVR de 8 bits fabricado por Atmel con los componentes complementarios suficientes para programarlo e incorporarlo a otros circuitos [26]. Un aspecto importante de Arduino es la manera estandarizada en la que sus conectores se colocan, permitiendo que la tarjeta principal pueda conectarse a una serie módulos de expansión, conocidos como escudos (inglés: *shields*). Algunos de estos escudos se comunican con el Arduino directamente mediante los pines de entrada/salida, además de ser individualmente direccionables mediante un bus serial que utiliza el protocolo I^2C, permitiendo que varios escudos puedan apilarse y ser utilizados en paralelo (Véase Figura 2.36 debajo).

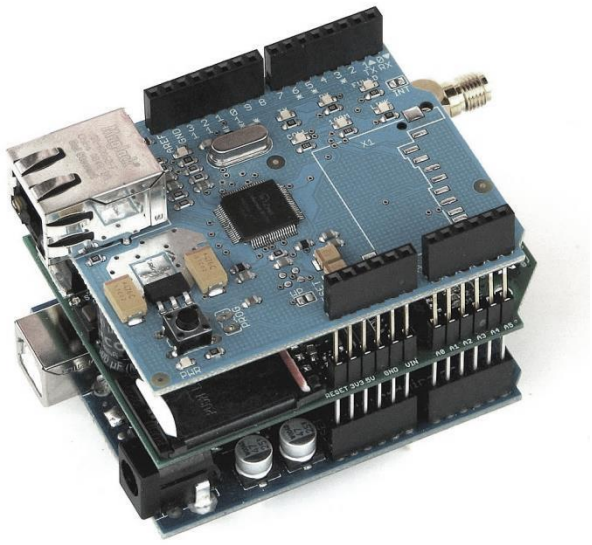


Fig. 2.36: Diferentes escudos apilados para utilizarse con una tarjeta Arduino.

Las tarjetas Arduino oficiales han utilizado la familia de chips megaAVR de Atmel, específicamente los Atmega8, Atmega168, Atmega328, Atmega1280 y Atmega2560. Unos cuantos procesadores de otros fabricantes han sido utilizados por tarjetas similares. La mayoría de las tarjetas incluye un regulador lineal a 5 volts y un oscilador de cristal de cuarzo a una frecuencia de 16 MHz (en algunas ocasiones reemplazado por un resonador cerámico) [26]

El microcontrolador del Arduino también viene pre-programado con un *boot loader* que simplifica el proceso de subir programas a la memoria flash del circuito, comparado con otras tarjetas que requieren un programador externo como accesorio para operar.

En un nivel conceptual, cuando se utiliza el software de programación de la plataforma Arduino, todas las tarjetas son programadas mediante una conexión serial de tipo RS-232, pero la forma en la que se implementa la conversión varía dependiendo de la tarjeta. Las tarjetas que poseen el conector serial tipo DB-9 utilizan un convertidor de niveles para pasar las señales de nivel RS-232 a nivel TTL. Las tarjetas Arduino actuales son programadas vía un puerto USB, implementado en la tarjeta mediante chips convertidores de USB-serial como el FTDI FT232. Algunos otros métodos de programación incluye utilizar accesorios tales como un programador bluetooth o un programador clásico de microcontroladores que se comunica mediante ISP.

La tarjeta Arduino expone la mayoría de los pines de entrada/salida del microcontrolador para poder utilizarlos con otros circuitos mediante headers. El Arduino UNO en específico provee (Figura 2.37):

- 14 pines digitales de entrada/salida, seis de los cuales pueden producir trenes de pulso tipo PWM.
- 6 pines analógicos utilizados como convertidores ADC.
- Pin de referencia para los convertidores analógicos.
- Pines de entrada/salida con voltajes de 5V, 3.3V y GND.
- Botón de reinicio maestro.
- Conector USB tipo B.

Fig. 2.37: Arduino UNO R3

Existe una gran variedad de tarjetas accesorio compatibles con Arduino de diferentes fabricantes. Algunas de estas vienen desensambladas con fines didácticos para estudiantes de electrónica, algunas traen componentes añadidos para expandir su funcionalidad, como por ejemplo *drivers* de motores DC para proyectos de robótica. Inclusive existen algunas tarjetas que utilizan otros procesadores con cierto nivel de compatibilidad entre escudos, tal como la Arduino Due la cual utiliza un ARM cortex con arquitectura de 32 bits, manteniendo la simplicidad de programación característica de la plataforma Arduino.

CAPÍTULO 3

DISEÑO DE LA PLATAFORMA

1. DISEÑO MECÁNICO

1.1. Introducción

Inicialmente, se planteó una plataforma la cual pudiese girar en tres grados de libertad mediante algún mecanismo como una articulación esférica u otro método. Esta plataforma poseería un sistema de compensación realista el cual podría ser probado directamente en la vida real. Obsérvese la Figura 3.1.

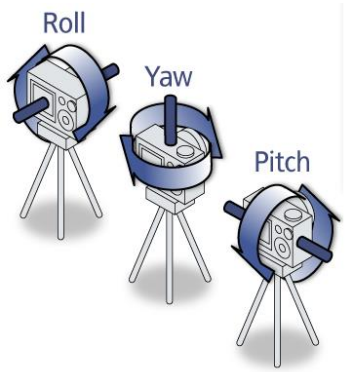


Fig. 3.1: Plataforma de tres grados de libertad.

Sin embargo, se optó por tomar un enfoque más sencillo en un inicio al hacer una plataforma con un solo grado de libertad. Al experimentar con esta plataforma y lograr control total sobre un grado de libertad, la expansión a dos o hasta tres grados será relativamente sencilla, siendo mucho más eficiente en términos de tiempo y costo.

La selección del eje de experimentación fue realizada para facilitar en general el desarrollo de los experimentos. Al construir una plataforma que girase ortogonal al eje z (esto es normal al suelo, con el lado positivo del eje apuntado hacia el azimut) la construcción, desarrollo y obtención de resultados sería la más eficiente, ya que la aceleración de la gravedad puede ser despreciada y en términos de velocidad angular la información obtenible es la misma en cualquier dirección de giro.

1.2. Plataforma principal

Ya que en este punto del diseño el tamaño y la forma de las articulaciones no fueron importantes, siempre y cuando entregaran una rotación confiable y constante, la selección de materiales y diseño mecánico en general se llevó tomando en cuenta características fundamentales, tales como:

- ✓ Robusto: Debe ser capaz de resistir los esfuerzos ejercidos por el motor.
- ✓ Ligero: Para que la carga en la flecha del motor sea mínima.
- ✓ Simple: La estructura más sencilla que pueda contener todos los componentes.

Para poder cubrir dichas características, la plataforma fue construida con materiales y piezas recicladas de otros proyectos en desuso, lo cual resultó en un costo neto casi nulo y un proceso de manufactura extremadamente sencillo. El hecho de que la plataforma experimental tuviese que girar alrededor de un solo grado de libertad fue una de las características que permitieron este acelerado proceso.

La única restricción que siempre se mantuvo en cuenta en el proceso de diseño y construcción fue la intercambiabilidad de partes (utilizando acoplamientos estandarizados y piezas desensamblables) y la interconectividad entre las mismas; estas cualidades nos permitirían cambiar piezas mecánicas o electrónicas rápidamente sin tener que invertir mucho tiempo o dinero. Una de las partes donde se puede apreciar mejor esta característica es en la tabla de pruebas o protoboard, donde se pueden probar los componentes sin necesidad de soldarlos. Una desventaja de utilizar dicha tabla de pruebas es la posibilidad de introducir ruido electrónico al momento de leer los sensores. Otra gran necesidad del proyecto fue la de incluir una pantalla LCD para poder bajar y revisar los datos obtenidos por los sensores de manera mas versátil.

Se puede observar un modelo CAD de la plataforma construida en la Figura 3.2. Se puede apreciar la cámara en la parte superior, la cual consta de un lente focal *Carl-Zeiss* con amplio ángulo de apertura, un chip *Silicon Retina* y un controlador FPGA para recibir y procesar la información. En la parte inferior trasera se observa al Arduino UNO y en la parte central una tabla de pruebas con los demás elementos electrónicos.

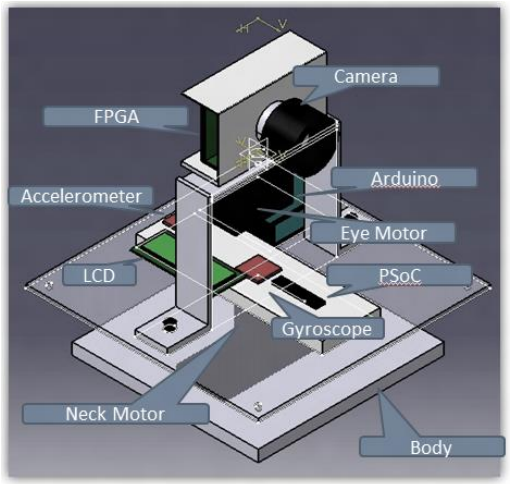


Fig. 3.2: Modelo CAD de la plataforma completa.[1]

1.3. Base Experimental

Debido a la necesidad de obtener datos de forma cualitativa, esto es, repitiendo el mismo experimento un número dado de veces, anotando los resultados y haciendo posteriormente un análisis para obtener un resultado, se divisó una base que pudiera hacer girar al resto de la plataforma a velocidad constante en ambos sentidos de giro. (Véase Figura 3.3).

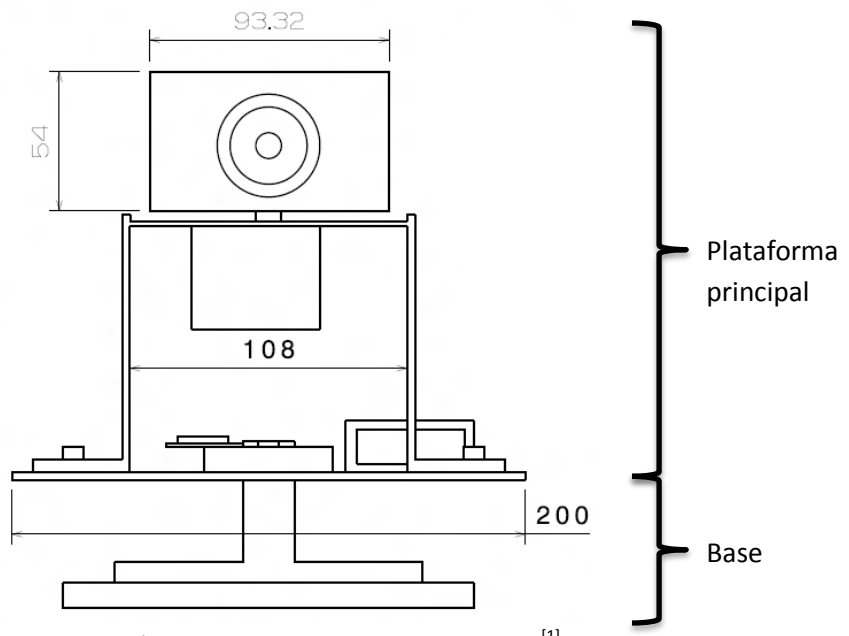


Fig. 3.3: Separación entre el cuerpo principal y la base.[1]

La única función de esta base es la de rotar al cuerpo principal (donde se encuentran los sensores y el resto de los componentes) en un rango de -90 a 90 grados tomando como referencia la posición inicial con el lente de la cámara viendo hacia el frente, esto es, a donde se encuentra el objetivo (un punto negro en un fondo blanco). Los parámetros de variación para los diferentes experimentos concluyeron como sigue:

- Aumentar o disminuir el rango de rotación.
- Aumentar o disminuir la velocidad de rotación.

Estas funciones se implementaron mediante controles analógicos conectados a otro microcontrolador PSoC completamente independiente de los de la plataforma principal, probando asi que no hay ningún tipo de interferencia entre ambos sistemas, sólo la reacción mecánica de uno con el otro.

Otros problemas que se observaron y corrigieron durante el proceso de construcción fue la necesidad de aumentar un soporte de un peso razonable para lograr que la plataforma inferior fuera completamente estática y todo el torque del motor fuera entregado a la plataforma superior, garantizando de esta forma que las mediciones de cada prueba no tuviesen que ser intervenidas manualmente.

2. Diseño Eléctrico

En términos de restricciones eléctricas, el sistema fue diseñado en un principio para operar con un voltaje de corriente directa de 5 V, con lo cual estaríamos garantizando la operatividad de las plataformas PSoC, Arduino y *Silicon Retina*. Sin embargo, el giroscopio ITG-3200, como ya se ha descrito en secciones anteriores, trabaja con un voltaja de operación de 3.3 VDC [14], por lo que se diseñó e implemento un convertidor de voltaje de 5 a 3.3 V (Figura 3.4), el cual funciona con un solo transistor, logrando el funcionamiento requerido con una topología simple. De esta manera conseguimos energizar todos los sistemas desde una sola fuente de 5V.

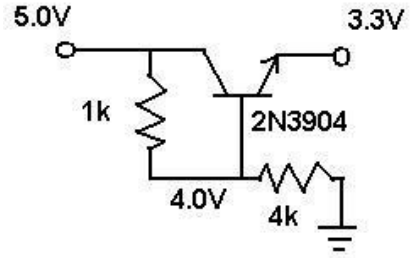


Fig. 3.4: Convertidor de voltaje [27].

El siguiente problema encontrado fue que la comunicación mediante el protocolo SPI funciona correctamente si alguno de los dos sistemas está alimentado a un voltaje diferente a 3.3V, por lo tanto todo el hardware fue reconfigurado para poder trabajar a 3.3 V. Sin embargo, el *display LCD* aún necesita los 5 V para funcionar adecuadamente, por lo cual se siguió utilizando una fuente conmutada de dos salidas.

Finalmente, cuando la señal de PWM era enviada desde el PSoC hacia los motores, con una diferencia de 3.3 V estos trenes de pulsos no eran reconocidas correctamente por el motor con la carga de la cámara acoplada en la flecha. Se implementó un conversor de voltaje discreto, el cual es capaz de aumentar o disminuir la amplitud de un tren de pulsos, utilizando el arreglo de transistores de la Figura 3.5.

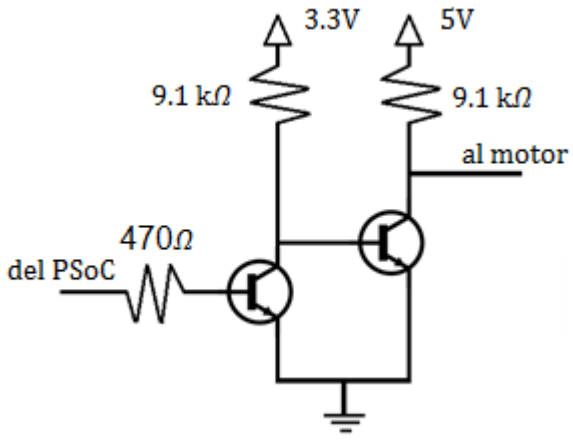


Fig. 3.5: Traductor de voltaje (de 3.3 a 5 V).

En la Figura 3.6 debajo se puede observar una fotografía de todas las conexiones del sistema (excluyendo a la plataforma Arduino UNO). Obsérvese el bus de 4 lineas para la conexión SPI entre FPGA y PSoC, y el bus de 2 lineas entre el ITG-3200 y el PSoC.

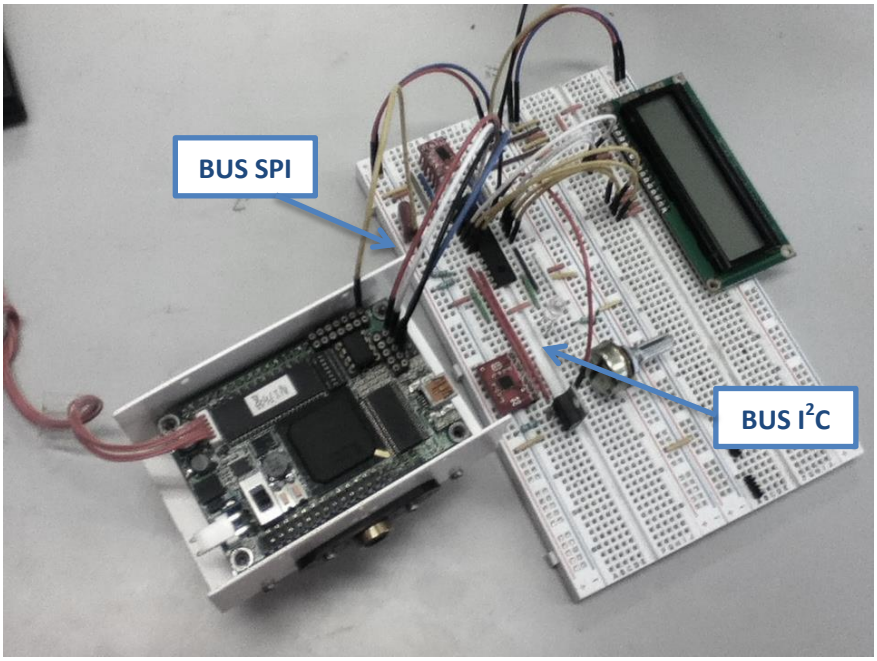


Fig. 3.6: Conexiones entre los diferentes elementos.

El diseño de la base experimental a nivel eléctrico /electrónico fue extremadamente sencillo. Se utilizó un segundo microcontrolador PSoC como apoyo, el cual convertiría las lecturas de un convertidor ADC provenientes de un potenciómetro en configuración de divisor de voltaje en pulsos PWM para alimentarlos al motor. Ya que todos los componentes operan a 5 VDC no hubo necesidad de implementar ningún tipo de convertidor. No se observaron fallos de operación por superar el torque de arranque del motor u otras anomalías.

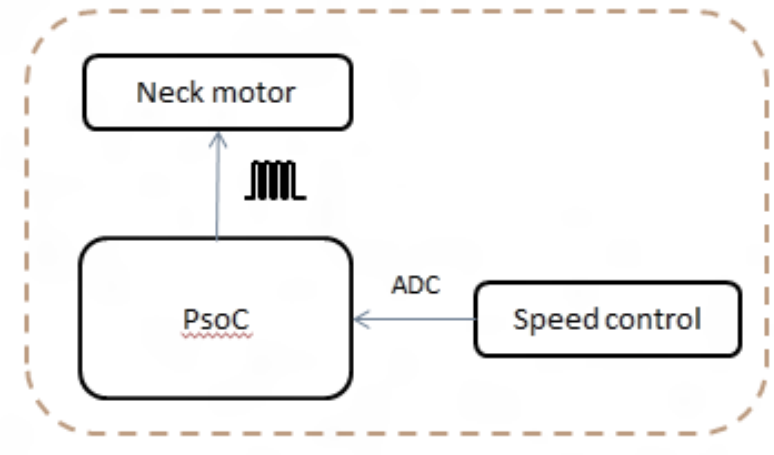


Fig. 3.7: Esquema eléctrico/electrónico de la base.

3. Diseño Electrónico

3.1. Transmisión de información

Inicialmente, la plataforma había sido diseñada para trabajar con dos sensores inerciales de 3 grados de libertad cada uno, logrando la capacidad de procesar hasta 6 variables para determinar la posición general del sistema. Sin embargo, al implementar la restricción a un solo grado de libertad, la información que es capaz de entregar el acelerómetro es de practicidad casi nula, ya que lo que el acelerómetro puede medir es la velocidad lineal en forma de perturbaciones externas, o bien el efecto del campo gravitatorio en la posición del sistema, valores que en este momento del diseño resultan inútiles. Por estas razones se determinó que de momento el acelerómetro quedaría fuera de los datos a tomar en cuenta aunque este siguiese colocado en la plataforma diseñada.

En un nivel electrónico, la plataforma principal consistió de:

- FPGA.
- PSoC.
- Arduino.
- Sensor visual (*Silicon Retina*).
- Sensor de velocidad angular (ITG-3200A).
- Servo motor.
- PC para procesamiento de datos.

La *Silicon Retina* es controlada por el FPGA utilizando un convertidor DAC y la retroalimentación de la cámara es a su vez recolectada utilizando convertidores ADC en paralelo de vuelta al FPGA. A pesar de que el FPGA tiene la capacidad suficiente para aplicar los filtros de procesamiento de imagen, en esta etapa de diseño se consideró prudente aplicar los filtros (Véase la Sección 3.2 de este Capítulo) mediante una librería de procesamiento de imagen (Open CV) y retransmitir la información al FPGA para ser utilizada por los demás dispositivos electrónicos. Esto es porque el lenguaje de programación y el tiempo que toma hacer tales filtros mediante Open CV es una fracción de lo que tomaría realizar las mismas acciones mediante FPGA. Ya que el algoritmo fue probado satisfactoriamente en la práctica, este puede transferirse a la memoria SRAM del FPGA.

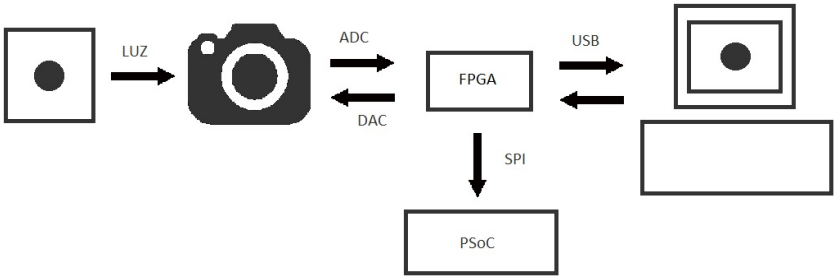


Fig. 3.8: Diagrama de conexiones para el procesamiento de imagen actual.

El FPGA está conectado al PSoC mediante el protocolo de comunicación SPI utilizando una topografía de un solo sentido, lo cual significa que el maestro siempre recibe datos del esclavo, o viceversa, sin necesidad de contestar ningún tipo de comando de reconocimiento (Para una breve descripción del protocolo SPI y sus funciones favor de referirse a la Sección 4, Capítulo 3). En este caso, el PSoC actúa como maestro y el FPGA como esclavo, y la única información que el FPGA envía es la posición en el eje x del centroide de la figura negra en un fondo blanco que tiene enfrente. Se puede observar un diagrama de conexiones en la Figura 3.8.

Cuando la información de la imagen de la cámara es recibida por el microcontrolador PSoC, éste la procesa para convertirla un un valor correspondiente al ángulo que debe moverse la flecha del motor, el cual es alimentado a un sistema de control de lazo cerrado tipo PI. El controlador PI será explicado con detalle en el Capítulo 4.

Al mismo tiempo, el giroscopio digital ITG-3200A está obteniendo información acerca de la velocidad angular de la plataforma con respecto a tres ejes ortogonales, transformando valores analógicos a digitales por si mismo y transmitiéndolos posteriormente al PSoC mediante el protocolo I^2C para ser procesados por el algoritmo diseñado (para más información acerca del protocolo I^2C referirse a la Sección 4, Capítulo 3).

El PSoC entonces procesa la información de ambos sensores, realiza las operaciones matemáticas necesarias y realiza las siguientes acciones de manera simultánea:

- Enviar información relevante acerca de los datos obtenidos al display LCD
- Mover al motor de la cámara de acuerdo al resultado obtenido
- Enviar toda la información que procesó al Arduino para ser guardada en una memoria SD.

Se puede observar un esquema de los componentes y sus conexiones eléctricas/electrónicas en la Figura 3.9 debajo:

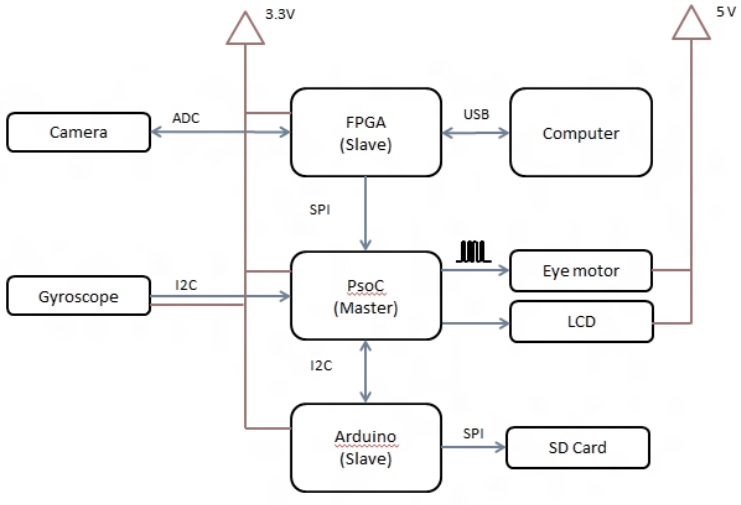


Fig. 3.9: Esquema eléctrico/electrónico del sistema.

Finalmente, en la Figura 3.10 a continuación se observa el *pinout* (la ubicación física de los pines) del chip utilizado, indicando la función asignada a cada uno.

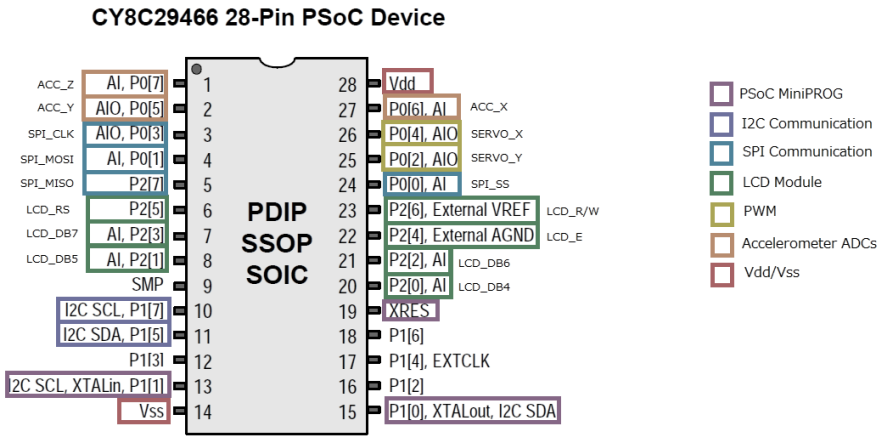


Fig. 3.10: *Pinout* del PSOC CY8C29466. [20]

4. PROCESAMIENTO DE DATOS

4.1. Procesamiento de imágen

El algoritmo utilizado para encontrar el centroide de una figura de color uniforme está basados en la teoría de los filtros dobles tipo Canny [28]. Considérese una imagen de 128x128 pixeles como una matriz cuadrada de orden 128x128. Ya que la imagen obtenida se encuentra en escala de grises de 8 bits, cada elemento de la matriz puede tomar un valor entre 0 y 255, donde 0 corresponde a negro total y 255 a blanco total. Tómese como ejemplo la Figura 3.11 abajo, donde los valores están normalizados, es decir, el valor máximo es 1, azul total y el valor más bajo es 0, blanco total, todos los valores intermedios corresponderán a un tono de azul.

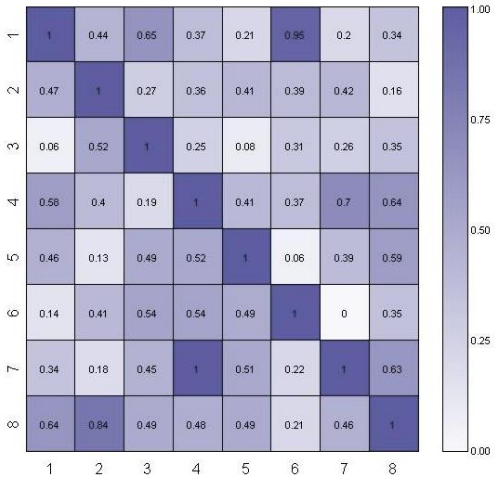


Fig 3.11: Matriz de pixeles de un color.

En estos filtros se compara cada pixel *P(i, j)* de la imagen con sus vecinos:

- En X: $P_{x+1}(i+1, j)$, $P_{x-1}(i-1, j)$,
- En Y: $P_{y+1}(i, j+1)$, $P_{y-1}(i, j-1)$.

Se establece un umbral de color, definido como:

$$\text{If } |Px(i, j)-Px+1(i+1, j)| > umbral \rightarrow Px(I,j) = 0$$
$$\text{If } |Px(i, j)-Px+1(i+1, j)| < umbral \rightarrow Px(I,j) = 255$$

Al realizar esta operación con los cuatro pixeles vecinos de cada pixel P, logramos obtener una matriz de valores binarios donde se resaltan solamente aquellos lugares donde hay un gran cambio de contraste, estos suelen ser los lugares pertenecientes al borde de un objeto con respecto al fondo, por ejemplo. El valor del umbral debe ser obtenido y calibrado experimentalmente para cada situación. Un ejemplo de la aplicación de este filtro tipo canny a una imagen puede observarse en la Figura 3.12:

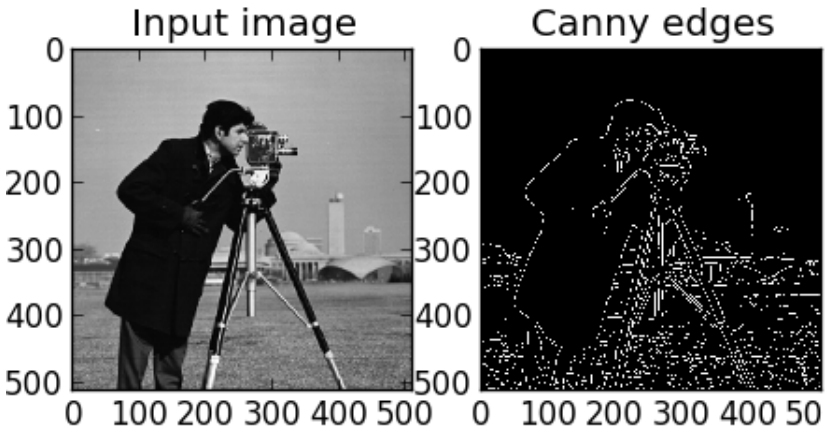


Fig. 3.12: Límites de los objetos encontrados utilizando un filtro Canny.

Posteriormente se realiza otro procedimiento mediante el cual se encuentra el centroide del objeto comparando todos los pixeles de cada fila y columna con los demás pixeles de su misma fila y columna [29]. Por ejemplo, al pixel P(1,1) se le compara con todos los pixeles de la fila 1 y de la columna 1. Al hacer esto se puede determinar mediante la densidad relativa de pixeles negros y blancos la posición del centroide mediante el siguiente algoritmo:

- Se cuenta el numero de pixeles que son negros
- Se divide este numero entre el numero de pixeles que hay en el eje x y en el eje y
- Estos numeros representan las coordenadas del centroide de la masa negra en fondo blanco.

Este procedimiento se repite por cada cuadro nuevo, es decir, al menos 30 veces por segundo. Un ejemplo del procesamiento que realiza el FPGA con ayuda de la PC se puede observar en la Figura 3.13.

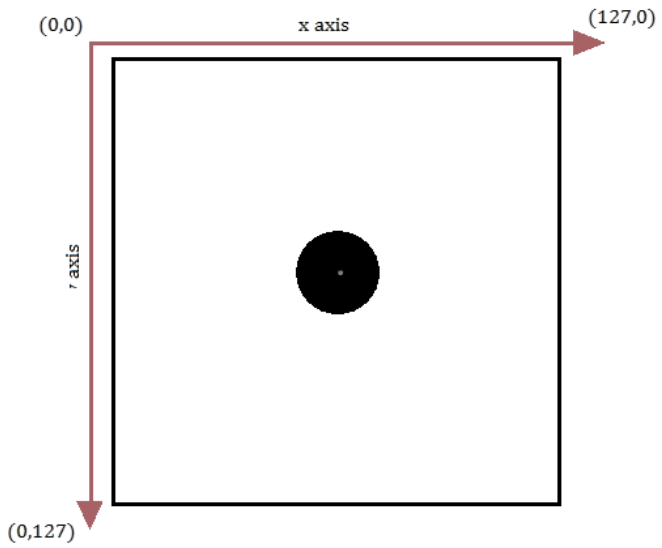


Fig. 3.13: Descripción de la imagen como es vista por el sensor visual.

Sólo el centroide de la masa negra es el dato que es enviado en un rango de 0 a 127 y es transformado por el PSoC al ángulo relativo al que se encuentra el punto en el plano de visión de la cámara, basado en esta información se genera otro dato que es insertado al controlador para calcular la posición a la que se debe de mover la flecha del motor para que el punto este exáctamente en el centro, esto es en el pixel P(0,64).

4.2. Lectura y almacenamiento de datos

Utilizando la plataforma Arduino se diseñó un sistema que grabara cada cierto tiempo los datos que recibe mediante el bus SPI de los otros dos microcontroladores que a la vez están procesando la información del giroscopio y la *Silicon Retina*. Un diagrama de flujo de información se puede observar en la Figura 3.14 como sigue:

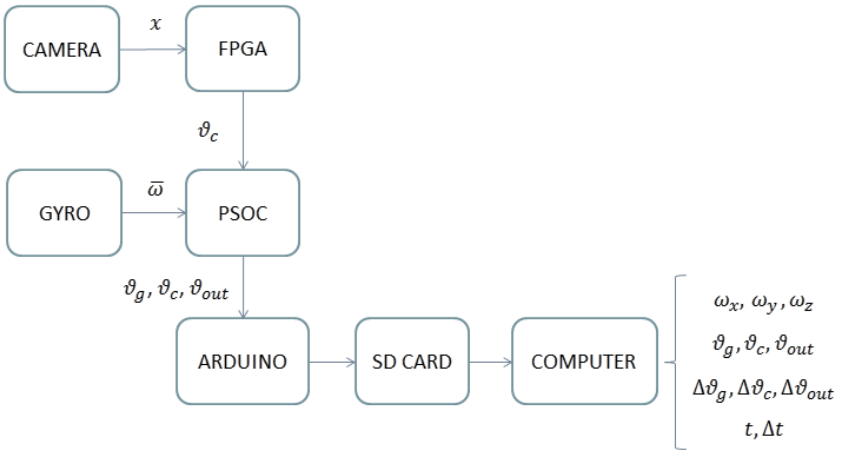


Fig. 3.14: Diagrama de flujo del *Logger*.

Los datos son grabados en forma de un archivo de hoja de datos en una memoria de almacenamiento masivo SD la cual podia ser utilizada posteriormente en una computadora para procesar los datos obtenidos, observar el comportamiento del sistema y generar modelos basados en dicha información. Un ejemplo puede observarse en la Figura 3.15:

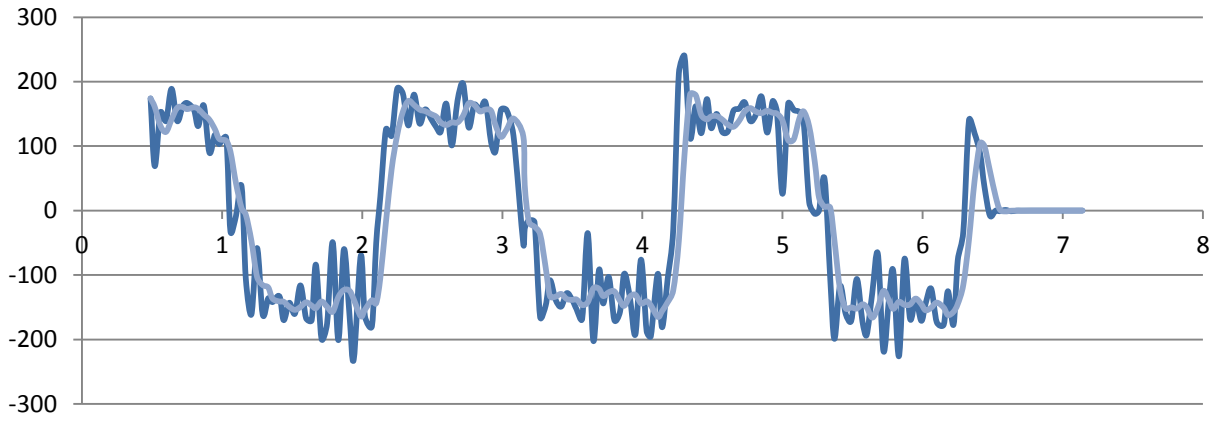


Fig. 3.15: Ejemplo de los datos obtenidos graficados en un procesador de hoja de cálculo.

5. PROTOCOLOS DE COMUNICACIÓN IC

5.1. Protocolo de comunicación SPI

SPI (Interfaz de Periféricos Seriales, inglés: *Serial Peripheral Interface*) es un estándar de comunicación por medio de un BUS serial establecido por Motorola y utilizado en productos semiconductores de muchos manufacturadores [30]. Las interfaces SPI están disponibles en los procesadores de comunicación más utilizados de marcas como *Texas Instruments, Microchip, Xilinx, Atmel,* etc. El tipo de conexión permite hacer una transmisión tipo duplex, es decir, puede transmitir y recibir señales simultáneamente mediante dos líneas independientes.

Los dispositivos se comunican entre si mediante una relación maestro/esclavo, en la cual el maestro es quien inicia la transmisión de datos. Cuando el maestro genera un pulso de reloj y selecciona al esclavo con el que desea comunicarse, la información puede transmitirse en cualquiera de los dos sentidos o en ambos a la vez. De hecho, de acuerdo a la especificación del protocolo SPI, siempre se están transfiriendo datos en ambas direcciones aunque estos no sean utilizados por los dispositivos, es responsabilidad de los mismos determinar si los datos son útiles o no. De modo que cada dispositivo debe:

- Descartar el byte recibido en el caso de una comunicación de tipo “sólo transmisión”, o bien,
- Generar un *dummy byte* para una comunicación de tipo “sólo recepción”.

Fig. 3.16: Implementación de un solo maestro y un solo esclavo en el protocolo SPI.

El protocolo SPI especifica cuatro señales que deben existir en todos los diseños que lo utilicen:

- El oscilador o reloj (SCLK, del inglés: *Souce Clock*).
- Salida de datos del maestro al esclavo (MOSI, del inglés: *Master-data-Out, Slave-data-In*).
- Salida de datos del esclavo al maestro (MISO, del inglés: *Master-data-In, Slave-data-Out*).
- Selector de esclavo (SS, del inglés: *Slave Select*).

La Figura 3.16 en la parte superior muestra estas señales en una configuración con un maestro y un esclavo. La señal SCLK es generada por el maestro y suministrada a todos los esclavos. La señal MOSI lleva los datos del maestro al esclavo, la señal MISO lleva los datos del esclavo de vuelta al maestro. El esclavo es seleccionado cuando el maestro pone el pin SS correspondiente al mismo en alto.

Si existen múltiples esclavos, el maestro debe generar una señal diferente para cada esclavo. Estas conexiones son ilustradas en la Figura 3.17:

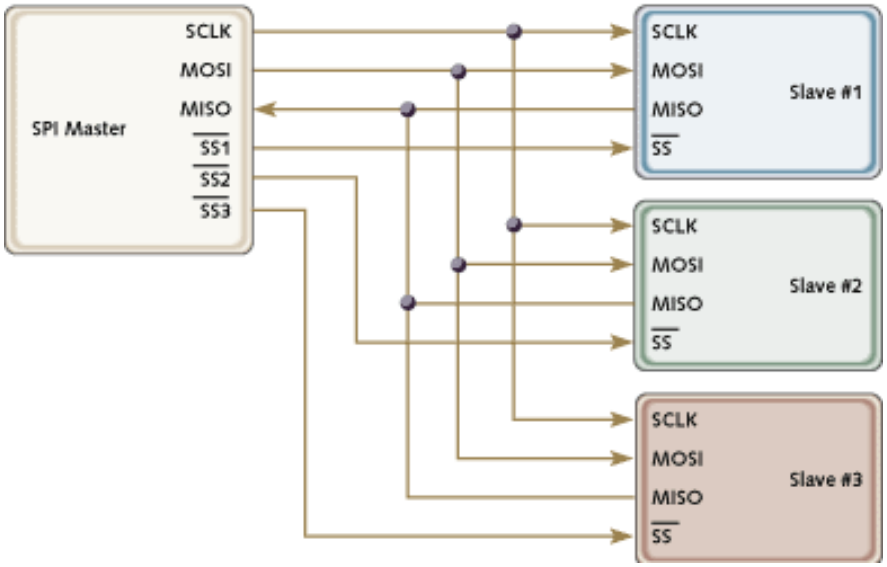


Fig. 3.17: Configuración de un maestro y múltiples esclavos en el protocolo SPI.

El maestro genera las señales de seleccion de esclavo utilizando pines digitales de entrada/salida de propósito general. Esto puede representar una limitación del protocolo cuando se cuenta con pines digitales

limitados y puede causar problemas si, por ejemplo, se cambia el direccionamiento del esclavo cuando una transacción está siendo realizada.

Aunque el protocolo SPI no describe una manera específica de implementar una topografía de tipo multi-maestro, existen dispositivos que ofrecen soporte para lograr este tipo de configuración. Sin embargo, es complicado y usualmente innecesario por lo que casi no se practica.

Existen dos parámetros llamados polaridad del reloj (CPOL, del inglés: *clock polarity*) y fase del reloj (CPHA, del inglés: *clock phase*) los cuales determinan los flancos de la señal del reloj en los cuales los datos se insertan en las respectivas líneas y en los que deben ser leídos por el dispositivo receptor. Cada uno de los dos parámetros tiene dos estados posibles, por lo tanto hay cuatro combinaciones posibles, todas de las cuales son incompatibles entre sí. Por lo tanto, un par de dispositivos maestro/esclavo deben utilizar los mismos parámetros para comunicarse satisfactoriamente. Si existen múltiples esclavos que tienen diferentes parámetros los cuales no pueden ser reconfigurados, el maestro deberá reconfigurarse a si mismo cada vez que necesite comunicarse con uno de estos esclavos.

El protocolo SPI también tiene ciertas desventajas, tales como el hecho de que no posee un mecanismo de reconocimientos de datos para determinar si los datos fueron recibidos o no. De hecho, sin un protocolo de comunicación el maestro SPI no tiene forma de saber si un esclavo existe o no. Tampoco se puede regular la velocidad del flujo de datos en este protocolo, de ser necesario para un proyecto en específico se deberá de buscar otras opciones.

Se puede pensar en los esclavos como dispositivos de entrada/salida pertenecientes al maestro. El protocolo SPI no especifica ningún tipo de regla para el diálogo entre el maestro y el esclavo. En muchas aplicaciones no es necesario construir una serie de reglas para transmitir/recibir información de modo que la información es transmitida en bruto, un ejemplo de esto puede ser un codec. En otras aplicaciones, se necesita implementar un protocolo de comunicación, tal como puede ser uno de tipo comando-respuesta.

Las capacidades de comunicación de tipo *full-duplex* y las velocidades de transferencia (las cuales pueden alcanzar el orden de varios megabits por segundo) hacen que la comunicación mediante el protocolo SPI sea, en la mayoría de los casos, extremadamente sencilla y eficiente. El protocolo se va complicando conforme el número de esclavos va aumentando debido a la falta de un direccionamiento pre-especificado, y puede llegar a ser ineficiente tener que configurar los cambios que debe realizar el maestro y alambrar una linea por cada esclavo que exista.

Lejos de ser un simple “puerto de bits”, el protocolo SPI ofrece una elegante solución para la mayoría de los proyectos que requieren sistemas de comunicación inter-circuitos. Además, también puede servir como base para desarrollar un protocolo más complejo si el proyecto lo requiere.

Nótese que el protocolo SPI es utilizado para la comunicación entre las plataformas PSoC y FPGA

5.2. Protocolo de comunicación I²C

I²C (Inter-Circuitos Integrados, inglés: *Inter-Integrated Circuit*) es una interfaz de dos hilos compuesta por las señales de datos serial (SDA, inglés: *serial data*) y reloj serial (SCL, inglés: *serial clock*). En general, las lineas son abiertas y bidireccionales. En una implementación generalizada del protocolo I²C, los dispositivos implicados pueden actuar como maestros o como esclavos. El dispositivo maestro escribe la dirección del esclavo en el bus, y el dispositivo esclavo con la dirección correspondiente reconoce al maestro. Hay modos de un solo maestro y multi-maestro incluídos en las especificaciones del protocolo I²C. Sin embargo, en este documento sólo se incluirá una explicación detallada de la configuración de un solo maestro, para más información acerca del protocolo I²C favor de referirse a [31].

Una configuración típica de un solo maestro en el protocolo I²C puede apreciarse en la Figura 3.18 a continuación:

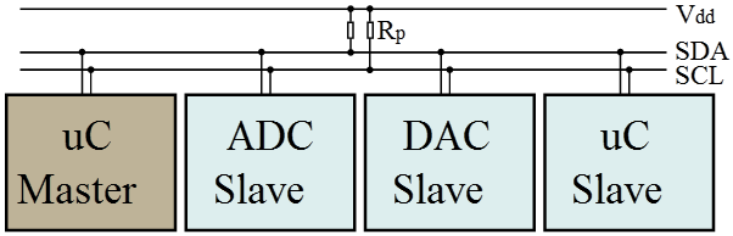


Fig. 3.18: Configuración tipica de maestro-esclavos en el protocolo I²C.

1. Condiciones de INICIO (S) y PARO (P)

La comunicación en el bus I²C comienza cuando el maestro pone la condición de INICIO (S) en el bus, la cual está definida como una transición de estado ALTO a BAJO de la linea SDA mientras la linea SCL está en estado ALTO*. Se considera que el bus está ocupado hasta que el maestro pone la condición de PARO (P) en el bus, que se define como un paso de BAJO a ALTO en la linea SDA mientras la linea SCL está en estado ALTO (Figura 3.19).

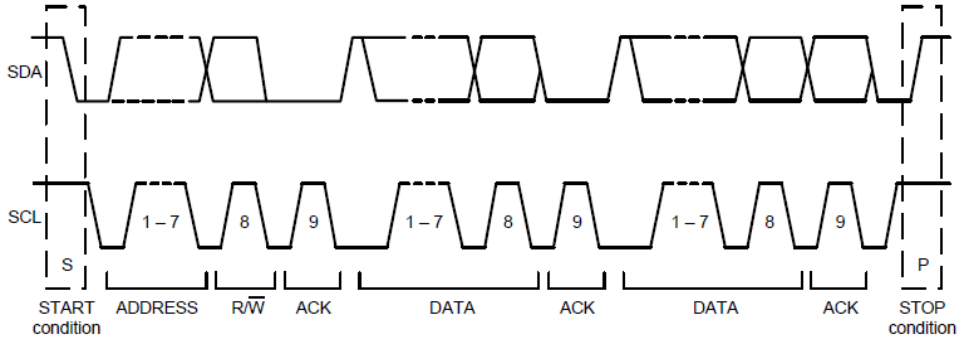


Fig. 3.19: Condiciones de inicio y paro en el bus I²C.

2. Formato de los datos y RECONOCIMIENTO (ACK)

Los bytes de información en I^2C están definidos para ser de 8 bits de largo. No hay restricción para el número de bytes que pueden ser transferidos en cada transmisión. Cada byte transferido debe ser seguido de una señal de RECONOCIMIENTO (ACK). El oscilador para la señal de reconocimiento es generado por el maestro, mientras que el receptor genera la propia señal bajando la linea SDA y manteniéndola abajo durante el pulso en ALTO del oscilador.

Si un esclavo está ocupado y no puede transmitir o recibir otro byte de datos hasta que otra tarea haya sido desempeñada, puede mantener la linea SCL baja, forzando al maestro a entrar en un estado de espera. La transferencia de datos normal continua cuando el esclavo está listo y libera la linea del oscilador (véase la Figura 3.20 a continuación):

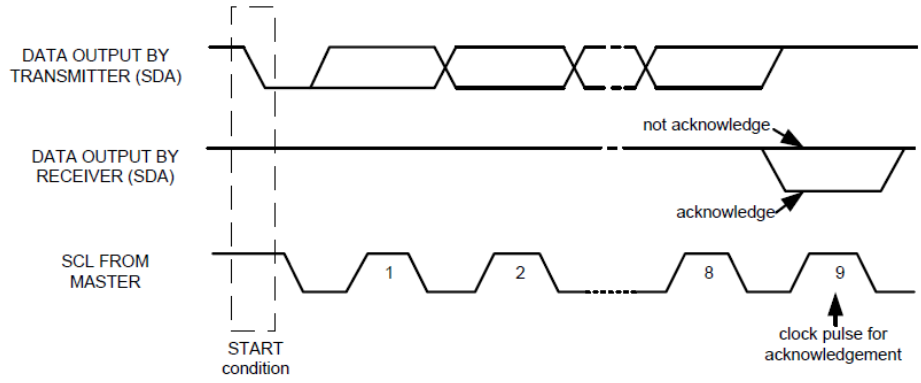


Fig. 3.20: Señal de reconocimiento del bus I^2C.

3. Comunicaciones

Después de haber comenzado la comunicación utilizando la condición de INICIO (S), el maestro envía la dirección de un esclavo de 7-bits seguido de un octavo bit, el bit de lectura/escritura. El bit de lectura/escritura indica si el maestro está recibiendo datos de o está escribiendo al dispositivo esclavo. Entonces, el maestro libera la linea SDA y espera la señal de RECONOCIMIENTO (ACK) del esclavo.

Cada byte transferido debe ser seguido por un bit de reconocimiento. Para efectuar el reconocimiento, el esclavo baja la linea SDA y la mantiene así durante el ciclo positivo de la linea SCL. La transmisión de datos siempre es terminada por el maestro mediante la condición de PARO (P), liberando la linea de comunicaciones. Sin embargo, el maestro puede generar una condición repetida de INICIO (Sr), y llamar a otro esclavo sin tener que generar una condición de PARO (P). Un paso de BAJO a ALTO de la linea SDA mientras que la linea SCL está en ALTO define a la condición de paro. Todos los cambios a la linea SDA deben de tomar lugar cuando la linea SCL esta en BAJO, con excepción de las condiciones de inicio y paro.

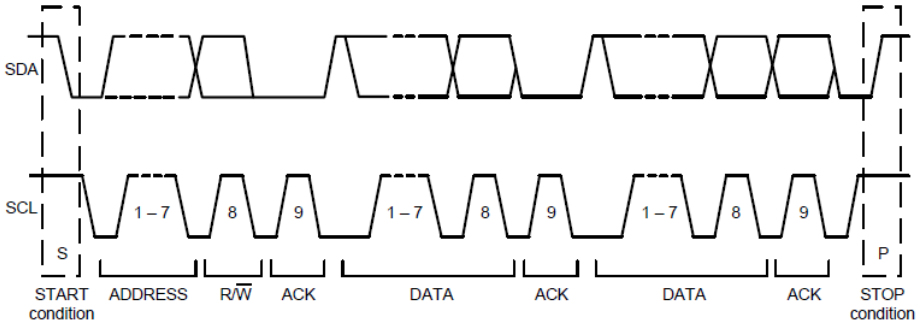


Fig. 3.21: Secuencia de transferencia de un byte en el protocolo I^2C.

El protocolo I^2C es relativamente nuevo y es una opción que cada vez mas manufacturadores de microcontroladores y otros dispositivos semiconductores están incluyendo en sus productos debido a la facilidad de implementación en hardware a cambio de una programación más robusta si se compara con, por ejemplo, el protocolo SPI explicado anteriormente.

El protocolo I^2C en este proyecto es utilizado para leer los datos del giroscopio, para comunicar a las plataformas Arduino y PSoC y para guardar los datos leídos en la memoria SD.

CAPÍTULO 4

SISTEMA DE CONTROL

1. INTRODUCCIÓN

1.1. Diagramas de control discreto [34]

Un sistema de control el cual utiliza una computadora como controlador o compensador es conocido como un sistema de control discreto. Las ventajas de utilizar computadoras digitales para sistemas de compensación incluyen: precisión, confiabilidad, economía y flexibilidad. El diagrama de bloques de un sistema de control se presenta en la Figura 4.1 debajo. En esta figura se encuentra encerrado en un cuadro punteado el controlador digital que consiste de una computadora y unos convertidores ADC y DAC (descritos en la figura con la nomenclatura A/D Y D/A, respectivamente), todos asociados a una misma señal de oscilador que los hace trabajar síncronamente. Nótese que la variable t cambia a T con una constante k y viceversa para representar el cambio de tiempo contínuo a tiempo discreto.

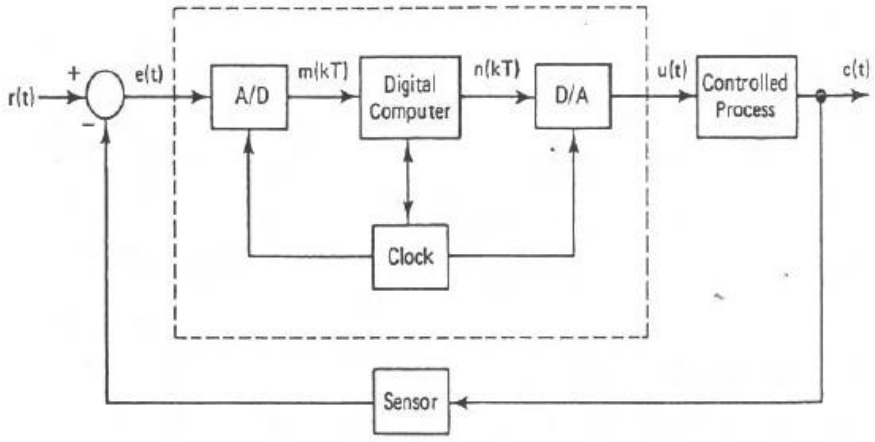


Fig. 4.1: Diagrama básico de un controlador digital de lazo cerrado.

Cabe mencionar que en la literatura no se suele representar de esta forma los sistemas de control discreto debido a la poca utilidad de estos elementos para el análisis del sistema. En vez de estos se utiliza solamente un muestreador.

Un muestreador (inglés: *sampler*) es un dispositivo que convierte una señal analógica en una señal digital (tren de pulsos) mientras que un dispositivo de espera (inglés: *hold*) convierte esta señal de pulsos en una señal contínua al mantener el valor obtenido hasta que el valor siguiente sea leído. Nótese que esto no es más que una abstracción de lo que es un convertidor ADC de un polo. El proceso se puede observar en la Figura 4.2:

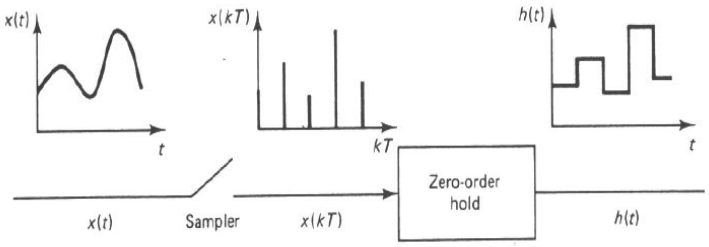


Fig. 4.2: Conversión de una señal continua a una señal discreta y su notación correspondiente.

En los sistemas de control digital clásico, se asume que todas las señales son muestreadas en intervalos regulares, síncronos y equitativamente distribuidos a través del tiempo, siendo T el periodo de muestreo, pero este no es el caso en la mayoría de las aplicaciones industriales de los sistemas de control.

Existen muchas razones para tener diferentes esquemas de muestreo.[33] Por ejemplo, la señal de control puede ser actualizada con una frecuencia constante pero los componentes del vector de resultados son medidas por diferentes sensores los cuales tienen diferentes tiempos de muestreo, características de ruido y confiabilidad. En algunos casos prácticos la salida del sistema no está disponible en cada instante de muestreo. Otra situación interesante puede resultar de sistemas de control con lazos múltiples, en los cuales las velocidades de muestreo son inclusive de orden diferente y ni siquiera están sincronizadas, este es el caso del presente proyecto.

Para este tipo de problemáticas existen los esquemas de control de velocidades múltiples, los cuales nos permiten seleccionar la velocidad de muestreo de acuerdo a las necesidades de la plataforma. Para esto, se requiere del uso de modelos de sistemas de control de tiempo discreto para cada lazo del sistema.

Giroscopio
Frecuencia de muestreo: 8 kHz
Intervalo de muestreo: 0.125ms

Silicon Retina
Frecuencia de muestreo: 30 Hz
Intervalo de muestreo: 33 ms

Fig. 4.3: Tiempos de muestreo de los sensores utilizados en el proyecto.

Como podemos observar en la Figura 4.3 arriba, los sensores utilizados en este proyecto tienen frecuencias de muestreo en ordenes de magnitud completamente diferentes, lo cual provoca la necesidad de construir un algoritmo especial mediante el cual se puedan utilizar ambos valores de la manera más eficiente posible para obtener el resultado combinado más preciso.

1.2. Transformada-Z

La Transformada-Z es una transformación lineal que se emplea entre otras aplicaciones en el estudio del Procesamiento de Señales Digitales, como son el análisis y proyecto de Circuitos Digitales, los Sistemas de Radar o Telecomunicaciones y especialmente los Sistemas de Control de Procesos por computadoras.

La Transformada-Z es un ejemplo más de Transformada, como lo son la Transformada de Fourier para el caso de tiempo discreto y la Transformada de Laplace para el caso del tiempo continuo. La importancia del modelo de la Transformada Z radica en que permite cambiar ecuaciones diferenciales a ecuaciones en diferencias o ecuaciones recursivas con coeficientes constantes a Ecuaciones Algebráicas lineales, las cuales son frecuentemente más sencillas de resolver para sistemas computacionales de baja potencia.

Esta transformada ha sido utilizada desde 1970 en distintos problemas de tiempo discreto y todas sus propiedades y características están bien definidas en la literatura matemática. Para leer más sobre la Transformada-Z se recomienda consultar la referencia [35].

1.3. Controladores PID discretos [36]

Los controladores Proporcional-Integral-Derivativos (PID) siguen siendo ampliamente utilizados en la industria por su simplicidad. Dependiendo del método de diseño, no es necesario modelar la planta, ni diseñar en específico sobre ese modelo. El usuario simplemente instala un controlador y ajusta las 3 ganancias para obtener el mejor desempeño posible. La mayoría de los controladores hoy en día son digitales.

Una ecuación estándar para un controlador PID puede ser:

$$\mathrm{u(t)} = \mathrm{K(e(t)} + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{dt}) \qquad [4.1]$$

Donde el error e(t), la diferencia entre el valor deseado y la salida de la planta, es la entrada del controlador y la variable de control u(t) es la salida del controlador (Véase Figura 4.4). Los tres parámetros son K (ganancia proporcional), Ti (tiempo de integración) y Td (tiempo de derivación).

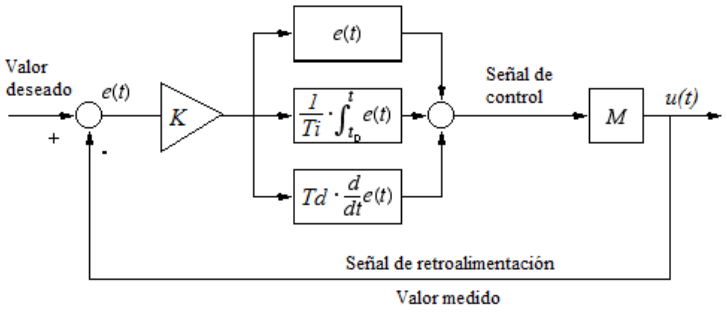


Fig. 4.4: Diagrama de bloques de un controlador PID

Haciendo la transformada de Laplace en la Ecuación 4.1 obtenemos:

$$\mathrm{G(s)} = \mathrm{K}(1 + \frac{1}{sT_i} + \mathrm{s}T_d) \quad [4.2]$$

Otra ecuación que describe a un controlador PID que también sera discutida en este documento se presenta en la Ecuación 4.3, esta forma es a veces llamada por algunos autores la forma paralela:

$$\mathrm{u(t)} = \mathrm{K}_p\mathrm{e(t)} + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}) \quad [4.3]$$

Y su transformada de Laplace:

$$\mathrm{G(s)} = K_p + \frac{K_i}{s} + \mathrm{s}K_d \quad [4.4]$$

Nótese que se puede pasar de una forma a otra simplemente aplicando las siguientes conversiones a las constantes:

$$\begin{aligned} K_p &= K \\ K_i &= \frac{K}{T_i} \\ K_d &= KT_d \end{aligned} \quad [4.5]$$

Sin embargo, para la implementación digital es de mayor interés la transformada Z de la Ecuación 4.3:

$$\mathrm{U(z)} = \left[K_p + \frac{K_i}{1 - z^{-1}} + K_d(1 - z^{-1})\right] E(z) \quad [4.6]$$

Reordenando:

$$\mathrm{U(z)} = \left[\frac{\left(K_p + K_i + K_d\right) + \left(-K_p - 2K_d\right)z^{-1} + K_d z^{-2}}{1 - z^{-1}}\right] E(z) \quad [4.7]$$

Se define:

$$\begin{aligned} K_1 &= K_p + K_i + K_d \\ K_2 &= -K_p - 2K_d \\ K_3 &= K_d \end{aligned} \quad [4.8]$$

La Ecuación 4.7 tambien puede escribirse como:

$$\mathrm{U(z)} - z^{-1}\mathrm{U(z)} = [K_1 + K_2 z^{-1} + K_3 z^{-2}]E(z) \quad [4.9]$$

Lo cual puede devolverse al espacio del tiempo con la transformada Z inversa como:

$$\mathrm{u(k)} = \mathrm{u(k-1)} + K_1 e(k) + K_2 e(k-1) + K_3 e(k-2) \qquad [4.10]$$

Con el grupo de Ecuaciones 4.8 y la Ecuación 4.10 tenemos un sistema apto para ser implementado en un sistema digital de poco poder computacional. Dicha implementación será disctuida en detalle en la Sección 4 de este Capítulo.

2. DESCRIPCIÓN DEL SISTEMA

El objetivo de la plataforma actual es hacer que la cámara (ojo) siga al objetivo rotando en la dirección opuesta en la que la plataforma (cabeza) se está moviendo, manteniendo como referencia la base (tierra), imitando el movimiento natural que realiza el ojo cuando estamos viendo a un objeto específico y movemos la cabeza de lado a lado (Figura 4.5). Para lograr este objetivo se utilizarán un sensor de velocidad rotacional (giroscopio) integrando a través del tiempo para obtener la posición angular y un sensor visual para conocer la posición respecto a una referencia visual (*Silicon Retina*).

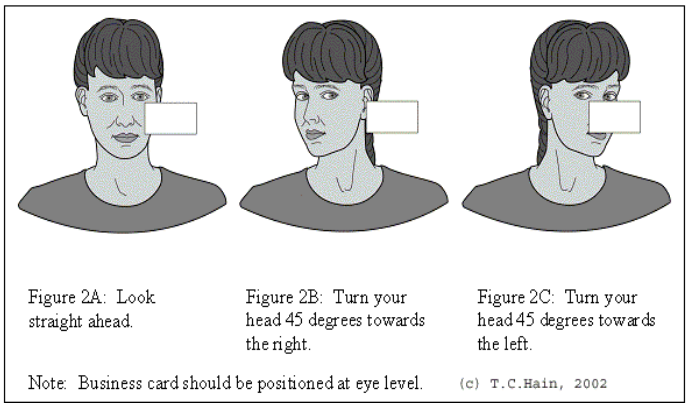


Fig. 4.5: Compensación mecánica de los ojos al girar la cabeza cuando se mira un objeto específico.

El arreglo de vectores que conforman a dicho sistema puede observarse en la Figura 4.6. Los ángulos a los que se debe prestar especial atención son:

- ✓ Θh/g: El ángulo entre la cabeza y la tierra, medido por el giroscopio.
  Este ángulo es medido directamente por el movimiento inercial del cuerpo.
- ✓ Θt/e: El ángulo entre el objetivo y el ojo, medido por la cámara.
  Este ángulo es medido indirectamente por la posición del objetivo respecto a la posición inicial o de tierra.
- ✓ Θe/h: El ángulo entre el ojo y la cabeza, el cual debe de hacerse cero por accion del motor.
  Idealmente este ángulo siempre debe tender a ser cero, **este es el objetivo general del sistema**.

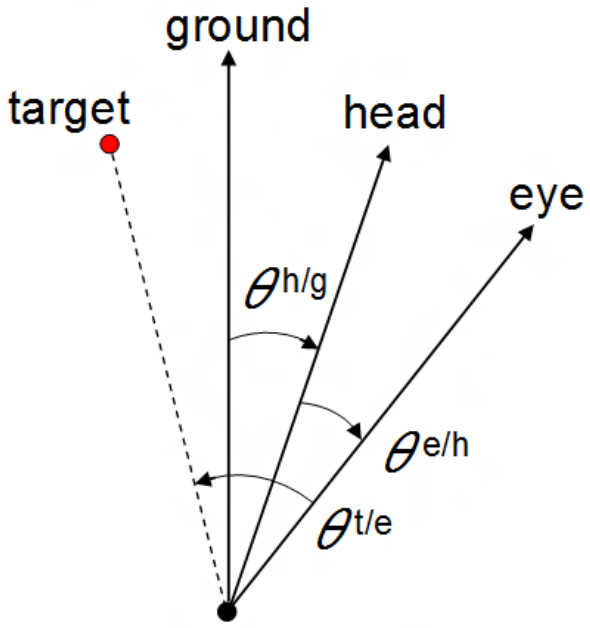


Fig. 4.6: Configuración de ángulos de la plataforma.

Estas relaciones pueden ser mejor observadas en la Figura 4.7. Los círculos rojos representan al rango móvil de los tres vectores previamente explicados (ojo, cabeza y tierra, nótese que el vector de tierra es una referencia y permanece constante) mientras que las flechas azules representan a los vectores angulares de cada sección de la plataforma en un tiempo dado *t*. En una posición inicial (*t* = 0) todos los vectores están alineados apuntando hacia el centro del objetivo.

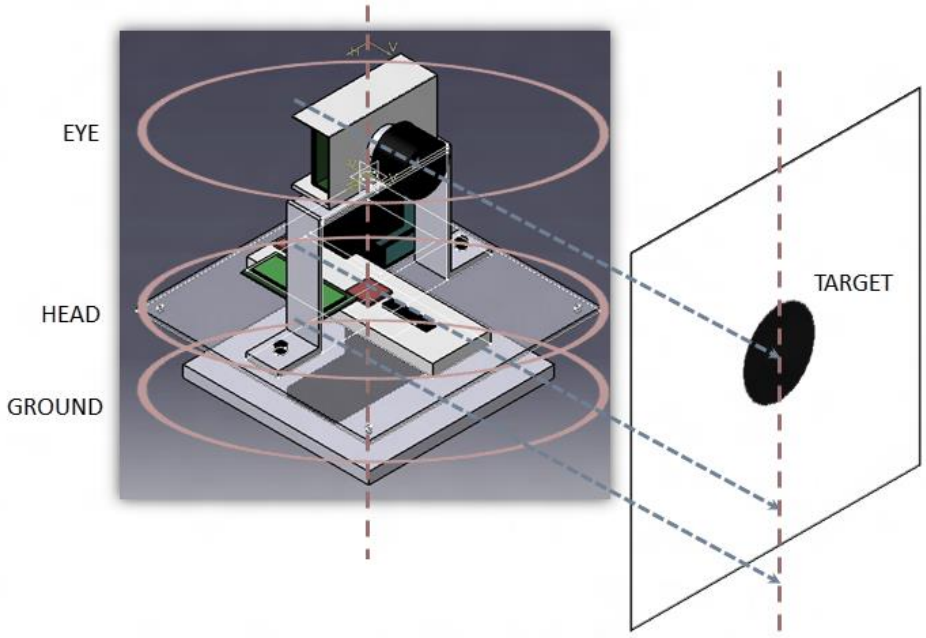


Fig. 4.7: Modelo 3D mostrando los vectores de la Figura 4.6.

3. CONTROLADORES PROPUESTOS

3.1. Control de acuerdo al giroscopio

Para cierto tiempo *t*, el giroscopio recibe como señal de entrada el ángulo de la cabeza con referencia a la tierra. Este ángulo es recolectado por el giroscopio ($\omega_{h/g}$), que al registrar las diferencias en el ángulo a través del tiempo actúa como un derivador. Posteriormente, éste ángulo, que es un valor contínuo, pasa por los bloques del muestreador para convertir el valor de la velocidad angular en discreto ($\omega_{h/g}$*).

Este valor es después integrado en el tiempo para poder obtener la variable de interés, el ángulo que se ha movido la cabeza con respecto a la tierra en cierto intervalo (1 [ms]), multiplicado por una ganancia k_1 para ajustar la escala a un valor de ancho de pulso que es alimentado al motor. Nótese que el mismo ángulo de entrada del giroscopio se suma a la salida del sistema para representar el movimiento de un estado anterior a un estado actual.

Al ser éste un controlador de lazo abierto, es decir, la salida del sistema no es medida directamente por el giroscopio y comparada con el valor deseado, presenta cierto error que se acumula a través del tiempo; este error es conocido como deslizamiento (inglés: *drift*) y será discutido a detalle en el Capítulo 5 del presente documento.

En la Figura 4.8 debajo podemos observar el diagrama de bloques que representa al control diseñado para el sistema tomando como referencia el valor que entrega el giroscopio:

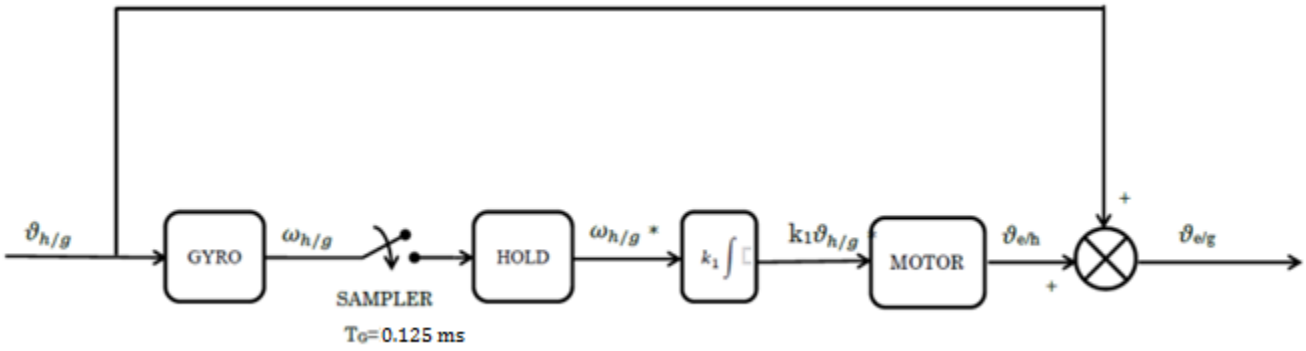


Fig. 4.8: Diagrama de control basado en los datos del giroscopio.

3.2. Control de acuerdo a la cámara (*Silicon Retina*)

La cámara es controlada utilizando un controlador de lazo cerrado tipo Proporcional-Integral (PI). Este controlador se implementó porque se observó que con un controlador proporcional la respuesta del sistema, al ser de primer orden, era demasiado lenta. Al agregar la parte integral se podía regular con más facilidad la respuesta del sistema discreto en términos de su tiempo de asentamiento y su oscilación.

En la Figura 4.9 debajo podemos apreciar el diagrama de bloques resultante para utilizar el *feedback* de la cámara. El valor deseado de este sistema es el ángulo entre el objetivo y el ojo, el cual debe tender a ser cero. Este valor se compara a la entrada con la lectura de la cámara y el resultado de esto es lo que se considera como valor de entrada del controlador PI. Obsérvese nuevamente el módulo discretizador justo a la salida del sensor (cámara).

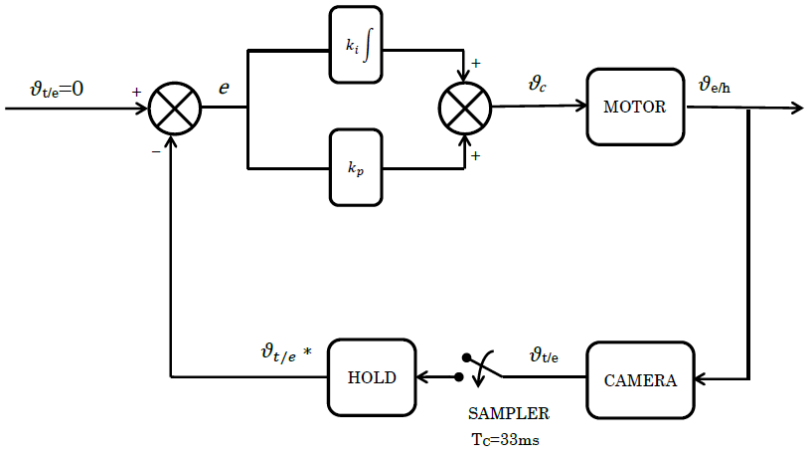


Fig. 4.9: Diagrama de control basado en los datos obtenidos de la cámara.

3.3. Control combinado

El diagrama de control completo combina los dos controladores presentados anteriormente, con la adición de un elemento para describir el algoritmo desarrollado para combinar de una manera efectiva los valores adquiridos por ambos sensores de acuerdo a sus frecuencias de muestreo individiuales.

Partimos de las siguientes dos consideraciones:

- El ángulo que entrega la cámara es muy cercano al ángulo real de la plataforma ya que al ser medido de acuerdo a una referencia externa la precisión del mismo es muy alta,
- El valor del giroscopio, al ser un valor medido por la velocidad intrínseca del sistema y ser un sistema de lazo abierto, presenta un error que se acumula a través del tiempo,

Llegamos a la conclusión de que nuestro valor referencia debe ser el de la cámara, mientras que el giroscopio actuará como mediador entre cada muestreo de la misma para entregar una resolución de movimiento mucho mayor sin sacrificar precisión.

Siguiendo esta linea de pensamiento se diseñó este bloque, visible en el centro de la Figura 4.10, el cual selecciona el tipo de operación que se debe hacer con la información de los sensores. El periodo del muestreador de la cámara es de 33 [ms], es decir, toma una muestra nueva cada 33 milisegundos, mientras que el periodo del muestreador del giroscopio es de 1 [ms], por lo tanto, por 32 milisegundos el valor obtenido por la cámara permanece constante.

El valor que sensa el giroscopio en el momento en el que la cámara toma su muestra es guardado en otro registro tipo *track/hold*, el cual está sincronizado con el muestreador de la cámara, de modo que cada vez que se actualiza el valor obtenido por la cámara, el valor obtenido por el giroscopio (el cual al ser instantáneo comparado con el de la cámara es de valor despreciable) es comparado con el de la cámara y el resultado de esta operación es alimentada al motor.

Las siguientes 32 veces que el giroscopio entrega un nuevo valor de posición angular (integrado a través del tiempo), este valor se suma al último valor conocido de la cámara, combinando de esta manera los datos de ambos sensores de manera efectiva de acuerdo a nuestra hipótesis.

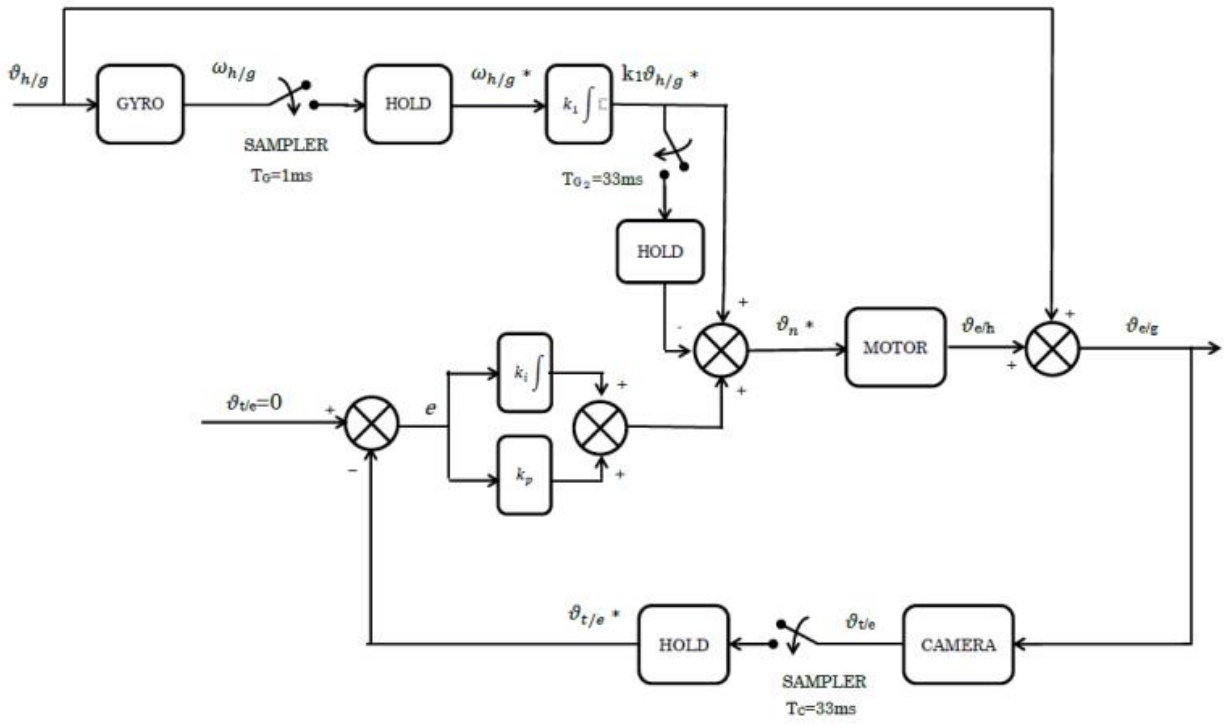


Fig. 4.10: Diagrama de control completo.

4. ALGORITMO

4.1. Ángulo de acuerdo al giroscopio

El siguiente algoritmo está basado en algunas ideas de los filtros Kalman [39], en específico en la comparación entre el estado anterior y el estado siguiente del sistema para predecir una salida más precisa. Sin embargo, el procedimiento está simplificado de manera que no requiere álgebra matricial, aligerando la carga computacional del CPU y por lo tanto, aumentando la capacidad del mismo.

Los datos del giroscopio son originalmente un valor analógico el cual es discretizado y transformado a un valor digital de 16-bits dentro del mismo chip y entregados al microcontrolador en esta presentación. Esta información es obtenida, como ya se había explicado anteriormente, mediante el protocolo de comunicación I^2C. Este valor debe ser procesado por el microcontrolador para convertirlo en un valor real.

Estos valores discretizados de 16 bits son representaciones proporcionales a los voltajes generados por el sensor. Para encontrar el voltaje correspondiente a cada uno de estos valores se utiliza la siguiente ecuación:

$$a_v = \frac{a_{16} v_{ref}}{step_{16}} \quad [4.11]$$

Donde a_{16} es el valor digital de 16 bits, v_{ref} es el voltaje de referencia del módulo ADC y $step_{16}$ es el número de valores posibles en un registro de 16 bits (un número de 16 bits tiene 2^{16} = 65,536 pasos de precisión). En el caso de un sensor analógico el voltaje equivalente de cierto valor discreto puede ser obtenido midiendo el valor del voltaje a la entrada del convertidor ADC y comparándolo con el resultado de la conversión del mismo.

Después, para transformar este voltaje a las unidades que el sensor registra, esto es la velocidad angular, medida en [deg/s], se puede aplicar la siguiente ecuación:

$$a_x = \frac{a_v - v_{zero}}{Sensibilidad} \quad [4.12]$$

Donde v_{zero} es el voltaje correspondiente al voltaje que entrega el sensor cuando su salida debe ser cero y la *sensibilidad*, usualmente expresada en [mV/unidad], es una constante que expresa la proporcionalidad entre el voltaje que entrega el sensor y las unidad que el sensor mide. Ambos valores pueden encontrarse en las hojas de especificaciones de cuaquier sensor comercial.

Al combinar las Ecuaciones 4.11 y 4.12 se halla la expresión que permite convertir el valor discreto de 16-bits a un valor expresado en [deg/s] para el giroscopio:

$$a_x = \frac{\frac{a_{16} v_{ref}}{step_{16}} - v_{zero}}{Sensibilidad} \quad [4.13]$$

Una vez que todas estas operaciones han sido realizadas es posible obtener la velocidad angular de cualquiera de los tres ejes y armar un vector de rotación. Sin embargo, para los alcances actuales del proyecto, solo requerimos saber la velocidad rotacional alrededor del eje z. Para encontrar la posición angular a partir del dato de la velocidad se puede utilizar la Ecuación 4.14:

[4.14]

$$\theta = \int_0^t \omega d\tau$$

Sin embargo, para la plataforma presentada en este trabajo es más conveniente trabajar con valores discretos, por lo tanto la ecuación de diferencias es necesaria (no la ecuación diferencial). Sabiendo que la velocidad es el cambio de posición a través del tiempo es posible escribir:

$$\omega(\mathrm{i}) = \frac{\theta(i) - \theta(i-1)}{\Delta t} \quad [4.15]$$

Despejando la variable de interés $\theta(i)$ de la Ecuación [4.15], obtenemos:

$$\theta(\mathrm{i}) = \omega(\mathrm{i})\Delta\mathrm{t} + \theta(\mathrm{i}-1) \quad [4.16]$$

En vez de utilizar solamente el valor actual de ω(i) se puede calcular el promedio del valor anterior y el valor actual mediante la Ecuación 4.17:

$$\omega_{est} = \frac{\omega(i) + \omega(i-1)}{2} \quad [4.17]$$

Al realizar el promedio entre el valor anterior y el valor actual se minimizan las variaciones entre los valores muestreados, minimizando el ruido presente en los datos analizados. El resultado de aplicar esta operación se puede observar en la Figura 4.11:

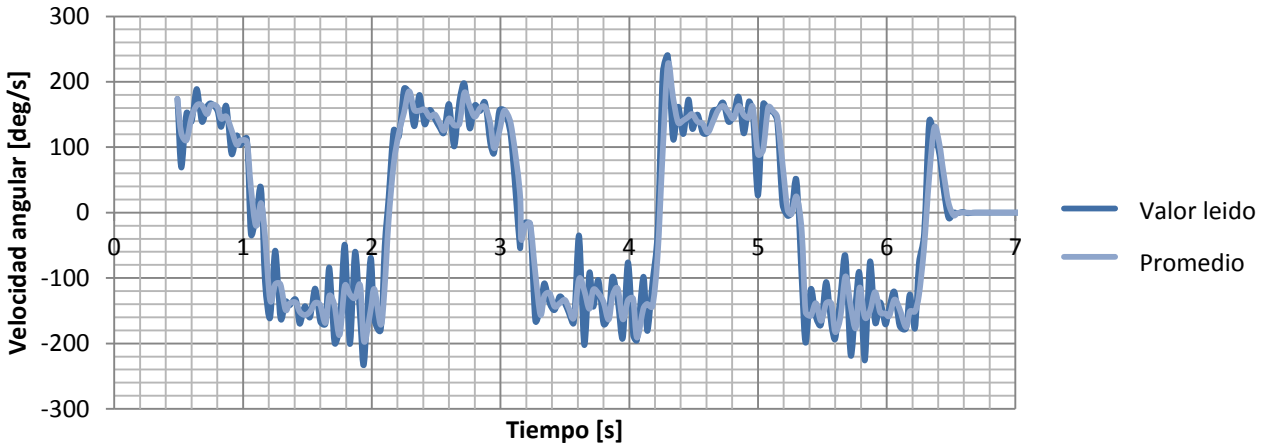


Fig. 4.11: Reducción de ruido utilizando el promedio del estado anterior y el actual.

Finalmente, sustituyendo la Ecuación 4.17 en la Ecuación 4.16 obtenemos:

$$\theta(\mathrm{i}) = \omega_{est}\Delta\mathrm{t} + \theta(\mathrm{i}-1) \quad [4.18]$$

Los resultados de aplicar la Ecuación 4.18 a los datos obtenidos del giroscopio pueden observarse en la Figura 4.12 debajo. Nótese que después de la integración discreta los datos forman una rampa relativamente constante a pesar del ruido que se encuentra en la señal original. Esta señal presenta otro problema que se conoce como deslizamiento [38] (inglés: *drift*) el cual fue medido experimentalmente y será discutido a detalle en el Capítulo 5.

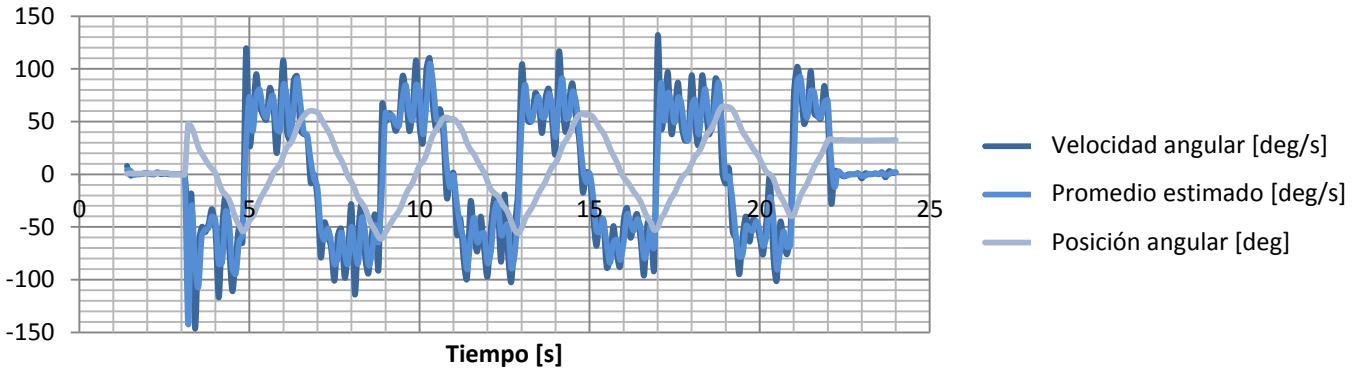


Fig. 4.12: Lectura original, resultado del promedio, integración en el tiempo

4.2. Ángulo de acuerdo a la cámara

En el caso de los datos recibidos por la cámara, como se ha explicado en secciones previas, la información recibida es una referencia visual externa, medida en pixeles (0 a 128) la cual es la entrada del controlador PI. Para explicar de manera adecuada el proceso que se lleva a cabo puede se puede observar en la Figura 4.13 el diagrama de flujo correspondiente.

Se definen las variables de entrada:

- Valor Deseado: Es el valor al que queremos que el controlador tienda. En este caso es 64, el valor intermedio entre 0 y 128.
- kp, ki: Son las constantes definidas para sintonizar el controlador tal como se explicó en la Sección 1.3 de éste Capítulo.

Las constantes k1 y k2 se definen mediante el sistema de ecuaciones [4.8] presentado previamente en este capítulo. También de la ecuación [4.10] se observa que es necesario para cada iteración conocer el error actual y el

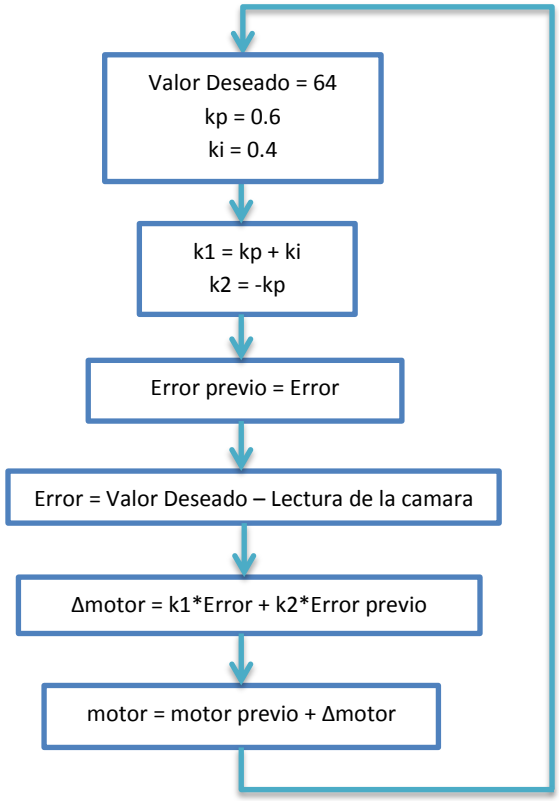


Fig. 4.13: Diagrama de flujo para controlar el motor de acuerdo al ángulo de la cámara.

error anterior, por lo quese guarda el error previo antes de calcular el error actual.

Posteriormente definimos el error actual como la diferencia entre el valor deseado y la lectura de la cámara. Estos dos valores son utilizados en la ecuación junto con los valores de k1 y k2 basándose nuevamente en la definición de la ecuación [4.10] y arroja como resultado el cambio que se debe de aplicar a la flecha del motor para compensar el movimiento y llegar al valor deseado. Finalmente, esta diferencia se suma al ángulo actual del motor para llegar al valor final del ángulo del motor.

El resultado de aplicar este algoritmo junto con un promedio del estado anterior y el actual como el demostrado para la información del giroscopio se puede apreciar en la Figura 4.14:

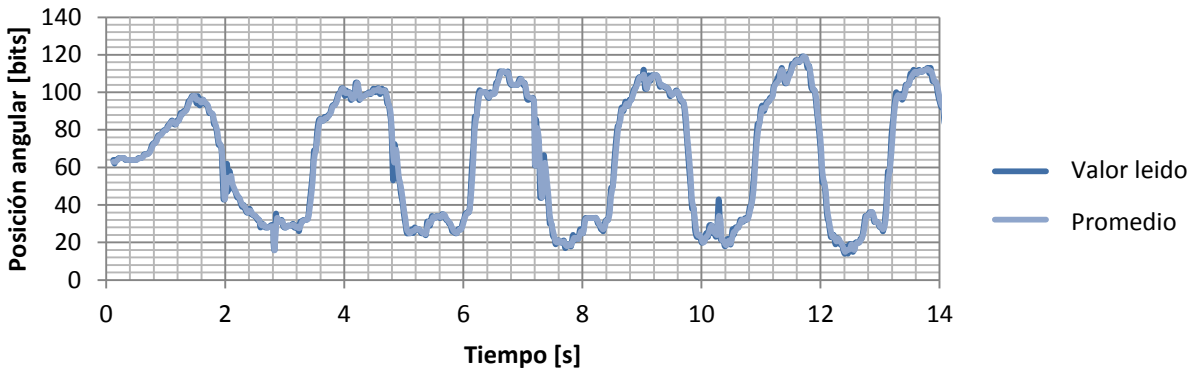


Fig. 4.14: Datos obtenidos de la cámara y el promedio del valor anterior y el valor actual.

4.3. Combinación de valores

Los valores obtenidos mediante los algortimos anteriores se combinaron de acuerdo al sistema propuesto en el diagrama de control mediante el algoritmo de la Figura 4.15:

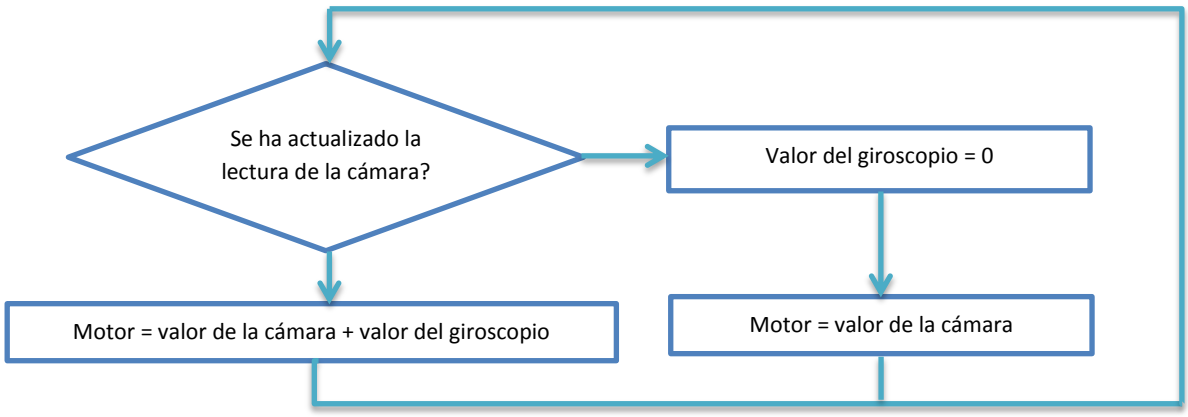


Fig. 4.15: Diagrama de flujo para combinación de valores.

El efecto de éste algoritmo se puede interpretar como sigue: “Si el valor muestreado por la cámara ya a sido actualizado, utiliza ese valor como salida del sistema. Si no ha sido actualizado aún, suma el valor que el giroscopio ha acumulado desde la última vez que se actualizó”. El código en lenguaje C correspondiente a los diagramas de flujo de las Figuras 4.13 y 4.15 se puede revisar en el Apéndice A del presente trabajo. Los resultados de aplicar este código experimentalmente serán mostrados y discutidos en el Capítulo 5.

CAPÍTULO 5

PRUEBAS REALIZADAS

1. DESPLAZAMIENTO (DRIFT)

1.1. Introducción

Los sensores inerciales, tales como el giroscopio y el acelerómetro, entre otros, presentan errores en sus lecturas inherentes a los mismos por una variedad de factores, entre estos se encuentran:

- El proceso de manufactura.
- El mecanismo mediante el cual el dispositivo sensa la variable de interés.
- La manera de interpretar estos datos para utilizarlos en la plataforma.

Este error es conocido como error de sesgo o desplazamiento [38] (inglés: *offset bias*). En el caso de los giroscopios, éste error se presenta cuando se requiere encontrar la posición angular de algún objeto calculada a partir de los datos obtenidos del giroscopio mediante un proceso de integración de la velocidad angular (en tiempo contínuo o discreto):

$$\theta = \int_0^t \omega d\tau$$

Al realizar esta operación, idealmente se debería de encontrar el ángulo exacto de la posición para cierto intervalo de tiempo; sin embargo, los giroscopios reales no son ideales y presentan ciertos niveles de precisión y de ruido, este ultimo es el que al momento de integrarse a través del tiempo genera un cálculo ligeramente erróneo (De acuerdo a las hoja de especificación del ITG-3200, el error oscila entre 0.1 y 0.01 del valor total medido), el cual puede ser despreciable para cierto intervalo de tiempo, más este error se acumula en cada nueva integración que se calcula. De este modo, después de cierto tiempo, el desplazamiento se hace notorio y los datos se vuelven inutilizables. Cabe resaltar que el error aumenta proporcionalmente a la velocidad que esta siendo medida por el giroscopio (Figura 5.1).

Es entonces necesario corregir el error acumulado en la medición del giroscopio cada cierto tiempo para que los datos permanezcan confiables para periodos de tiempo prolongados. Existen varios métodos para proveer de esta compensación a la lectura provista, en el caso de esta plataforma se utilizan los datos de la cámara (los cuales se consideran precisos) cada cierto intervalo de tiempo.

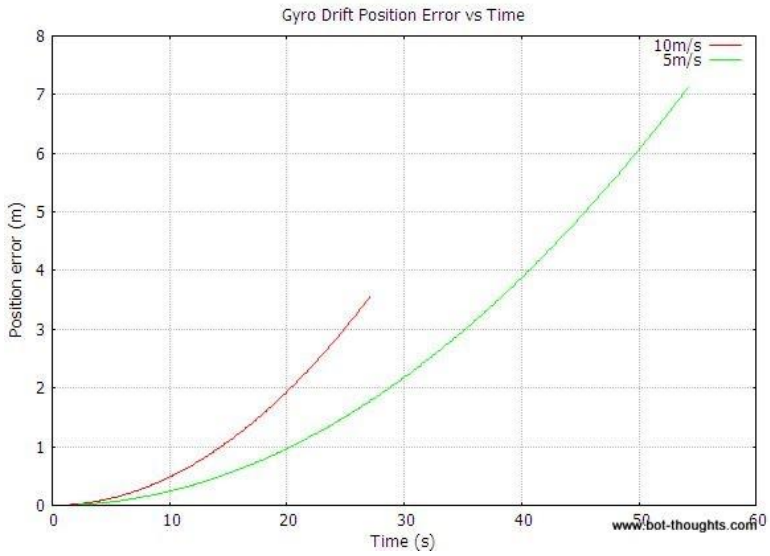


Figura 5.1: Error acumulado en las lecturas de un giroscopio.

Para poder estimar si el intervalo de actualización con los datos de la cámara es suficiente para que el error del giroscopio no provoque un comportamiento anómalo en el sistema se realizaron pruebas para determinar experimentalmente el valor de este sesgo. En la siguiente sección se presentan los datos obtenidos y un análisis de los mismos.

1.2. Descripción del experimento

Para obtener resultados que sean confiables de acuerdo a la metodología científica [7], se diseñó un experimento el cual cumpliese con las siguientes características:

- Solo una variable a la vez: Todos los demás parámetros deben permanecer constantes a fin de que no haya ruido en el sistema
- Repetibilidad: El mismo experimento debe poder llevarse a cabo dos o más veces con las mismas condiciones iniciales y se deberían observar resultados similares.

A continuación se presenta el algoritmo seguido para todas las pruebas realizadas:

1. **Posicionar** la cámara a 30 grados en dirección antihoraria a partir de la posición inicial:

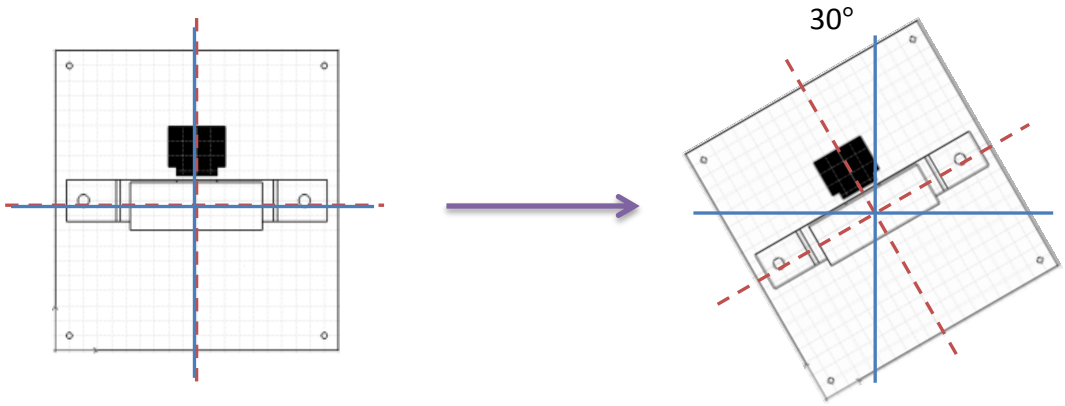


Fig. 5.2: Rotación inicial del sistema.

2. **Mover** a la plataforma 60 grados en sentido horario, posteriormente girar 60 grados en sentido antihorario hasta la posición original.

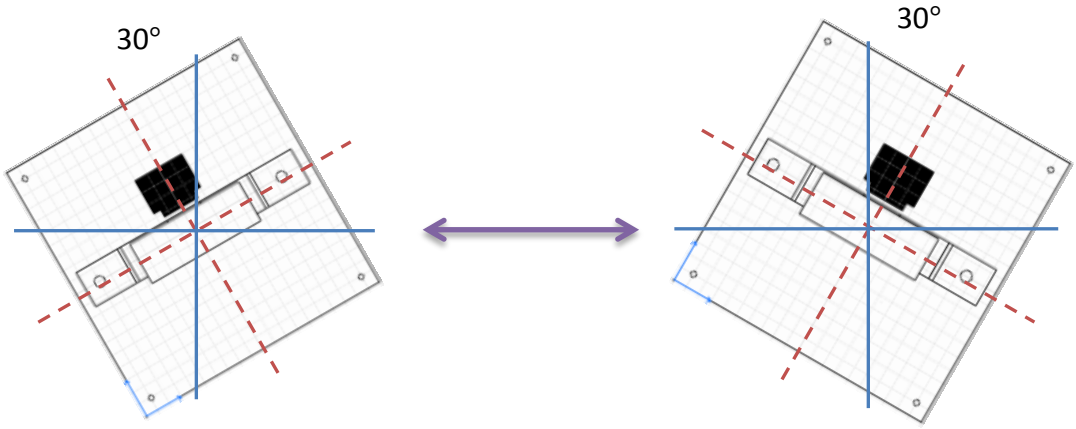


Fig. 5.3: Rotación cíclica.

3. **Repetir** el proceso hasta cumplir cierto número de ciclos predeterminado.

Este procedimiento sería repetido múltiples veces, siendo los valores controlables por el operador la velocidad de rotación y el número de ciclos que se realizarían en cada prueba.

1.3. Resultados

A continuación se presentan 3 juegos de pruebas realizadas a la plataforma. En estos la velocidad angular se mantiene constante y la variable de cambio es el número de ciclos que realiza la plataforma

1. **PRUEBA 1: 10 ciclos**

Prueba	tiempo [s]	Desplazamiento total [deg]	Velocidad de desplazamiento [deg/s]
1	24.5	10	0.408163265
2	23	15	0.652173913
3	24	13	0.541666667
4	24	18	0.75
5	24.5	14	0.571428571
6	24.18	13.2	0.545905707

Tabla 5.1: Resultados para la prueba 1.

- Tiempo promedio = 24.18 [s]
- Ángulo desplazado promedio = 13.2 [deg].
- Sesgo promedio = 0.5459 [deg/s]

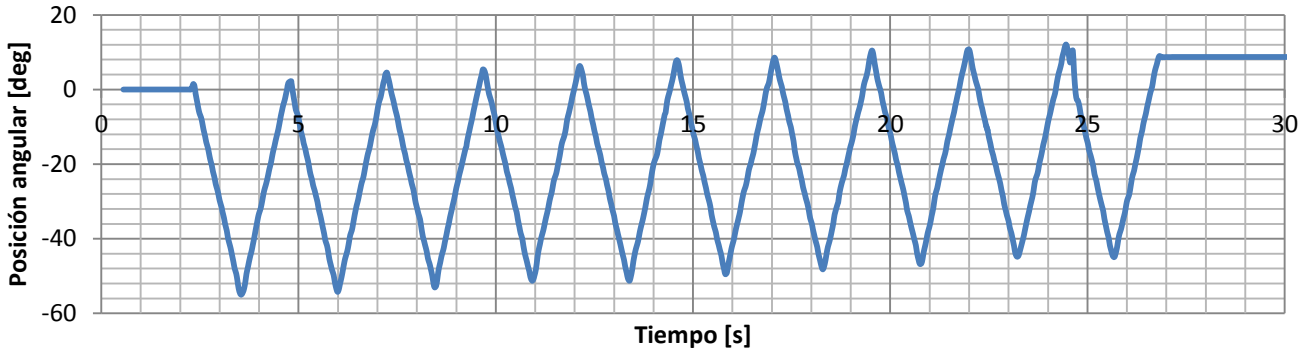


Fig. 5.4: Muestra al azar de la prueba 1.

2. **Prueba 2: 20 ciclos**

Prueba	tiempo [s]	Desplazamiento total [deg]	Velocidad de desplazamiento [deg/s]
1	50.2	20	0.3851
2	51	19.8	0.39
3	50.6	25	0.49
4	49.8	22	0.44
5	50	23.5	0.47
6	50.1	20.5	0.41

Tabla 5.2: Resultados para la prueba 2.

- Tiempo promedio = 49.1 [s]
- Desplazamiento angular promedio = 22 [deg]
- Sesgo promedio = 0.4483 [deg/s]

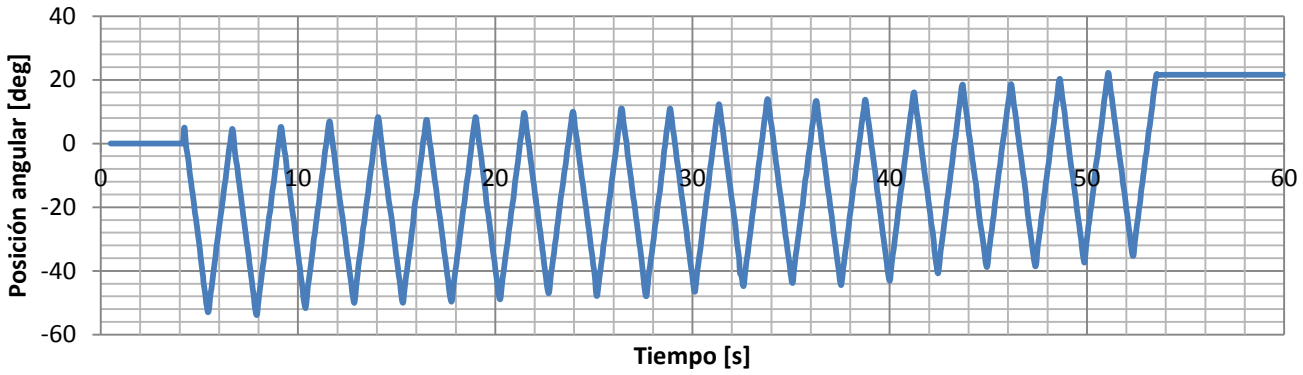


Fig 5.5: Muestra al azar de la prueba 2.

3. Prueba 3: 50 ciclos

Medición	tiempo [s]	Desplazamiento total [deg]	Velocidad de desplazamiento [deg/s]
1	125.8	68	0.540540541
2	125.9	52	0.413026211
3	126	76	0.603174603
4	125.8	58	0.461049285
5	125.9	54	0.428911835
avg	125.88	61.6	0.489340495

Tabla 5.3: Otros resultados de la prueba 3.

- Tiempo promedio = 125.88 [s]
- Desplazamiento angular promedio = 61.6 [deg]
- Sesgo promedio = 0.4893 [deg/s]

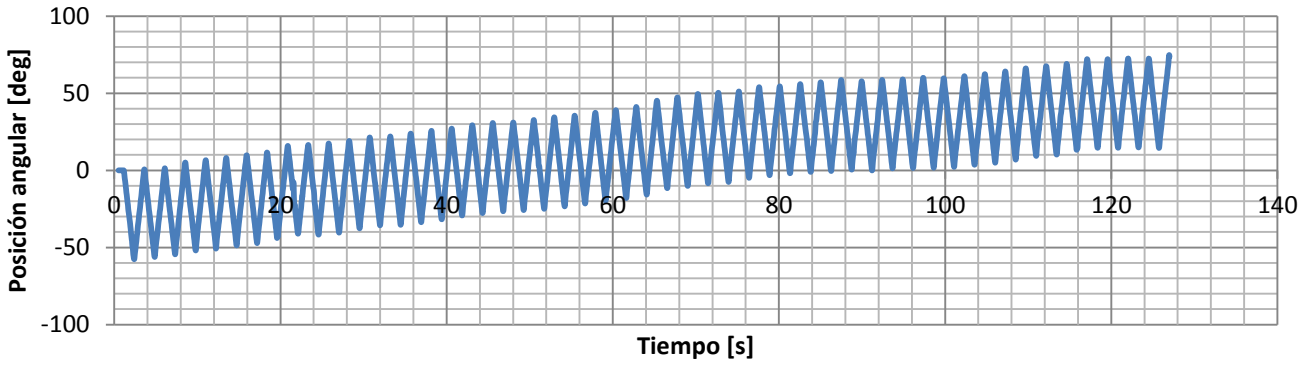


Fig. 5.6: Muestra al azar de la prueba 3.

Se observa que en las tres pruebas el valor promedio del desplazamiento del valor medido por el giroscopio en relación al valor real oscila alrededor de **0.5 [deg/s]**. Este valor se redondea ya que no es necesaria demasiada precisión debido a las cortas escalas de tiempo, como se analizará en el apartado siguiente.

2. RESULTADOS SIMULADOS

El algoritmo fue probado en primera instancia creando una simulación de los resultados esperados de los sensores. Dicha simulación fue creada basándose en los mismos parámetros utilizados para la prueba de desplazamiento, es decir, la plataforma giraría en un arco de 60 grados en ambos sentidos a velocidad constante alrededor del eje z.

En una primera iteración de dicha simulación podemos observar tres valores graficados:

- **Valor real:** Es el valor ideal que debe tener la plataforma de acuerdo a las condiciones establecidas. Los valores simulados de los sensores están basados en éste.

- **Giroscopio:** El ángulo que representa el valor del giroscopio fue programado para seguir al ángulo real presentando un aumento en el valor absoluto conforme pasa cada lectura. Aunque el valor experimental que encontramos para el desplazamiento que presenta el giroscopio es de 0.5 [deg/s] en la mayoría de las simulaciones utilizamos un valor de 10 [deg/s] para el sesgo para que el cambio sea apreciable a los pocos segundos de iniciar el algoritmo.

- **Cámara:** El valor de la cámara fue diseñado para seguir con una precisión total al valor real, con un retardo entre cada muestra de 33 milisegundos, simulando la baja frecuencia de muestreo de la *Silicon Retina*. Esto provoca un efecto escalera en la señal de la cámara como se aprecia en la Figura 5.7.

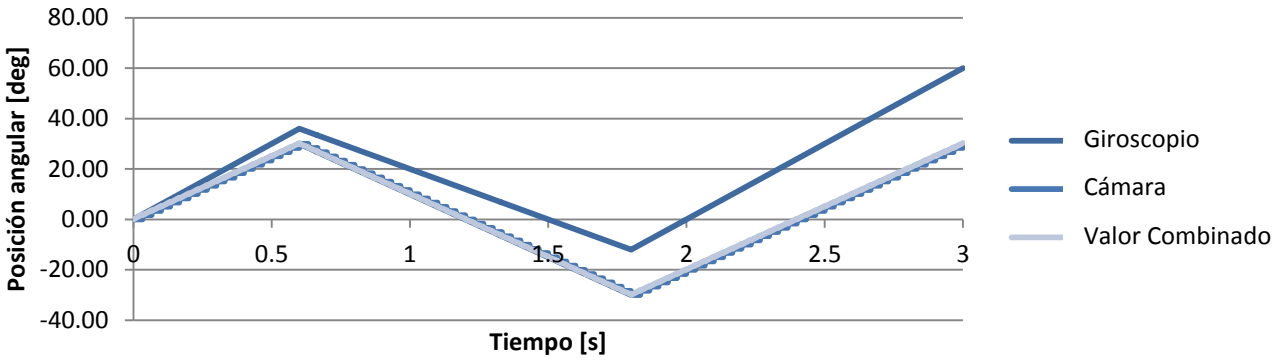


Fig. 5.7: Simulación del desplazamiento del giroscopio y de la cámara con alta latencia.

Al aplicar el algoritmo desarrollado, se combinan los valores de ambos sensores para producir la señal de salida que se observa en la Figura 5.8:

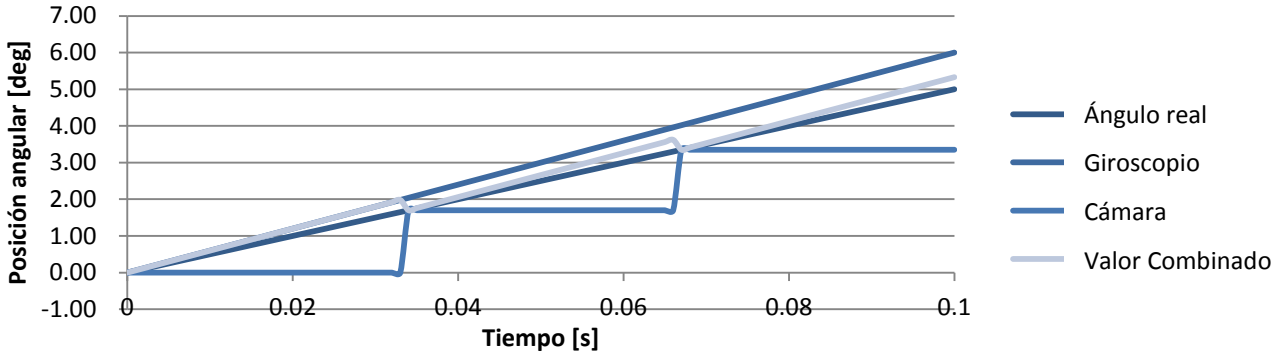


Fig. 5.8: Acercamiento de la Figura 5.7 se observan todos los ángulos de la figura anterior y el ángulo esperado.

En la Figura 5.8 arriba podemos observar a mayor detalle los primeros 1000 milisegundos de la simulación presentada en la Figura 5.7. Aquí se hace evidente la efectividad del algoritmo. La señal *est* (resultado estimado por el algoritmo) sigue a *gyro* (valor del giroscopio) hasta que *camera* (valor de la cámara) es actualizado 33 milisegundos después de haber iniciado la simulación. Este proceso se repite de modo que la desviación del valor real permanece en un rango despreciable, removiendo exitosamente el desplazamiento generado por el giroscopio.

Finalmente, en la Figura 5.9 se puede apreciar una simulación en un tiempo de 7 segundos utilizando el valor de desplazamiento obtenida en las pruebas realizadas al giroscopio. Como ya hemos determinado que el algoritmo funcionaría con un valor de desplazamiento de 10 [deg/s], esta simulación simplemente se realizó para reiterar que el algoritmo funcionaría también en condiciones reales.

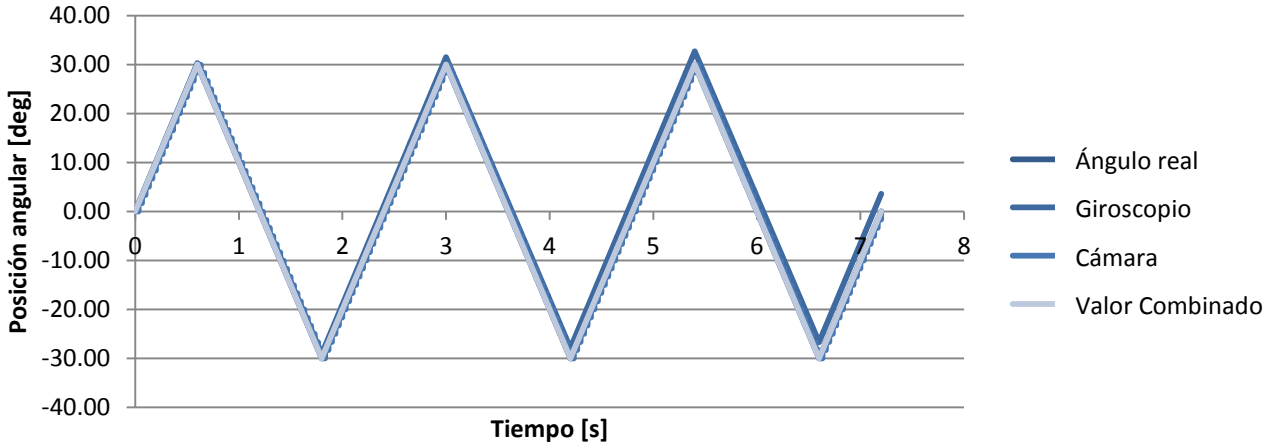


Fig. 5.9: Simulacion de los resultados del giroscopio utilzando un valor de arrastre real.

3. RESULTADOS EXPERIMENTALES

El algoritmo desarrollado y probado mediante simulaciones fue programado en la plataforma experimental. En las Figuras 5.9 y 5.10 se puede apreciar a todo el sistema ensamblado en el laboratorio de experimentación.

En la figura 5.10 se observan todos los elementos principales:

- **Plataforma principal:** Consta de la cámara (*Silicon Retina*, FPGA), un servomotor (motor de ojo) y la circuiteria (giroscopio, PSoC, Arduino)
- **Base de experimentación:** Justo debajo de la plataforma principal, consta de un servomotor (motor de cabeza) y el controlador (PSoC)
- **Objetivo:** Consta de un punto negro en un fondo blanco.
- **Computadora personal:** Utilizada para visualizar la salida de la cámara y realizar el procesamiento de imagen.
- **Fuente de voltaje:** Fuente de laboratorio estándar (0-30 V, 3 A)

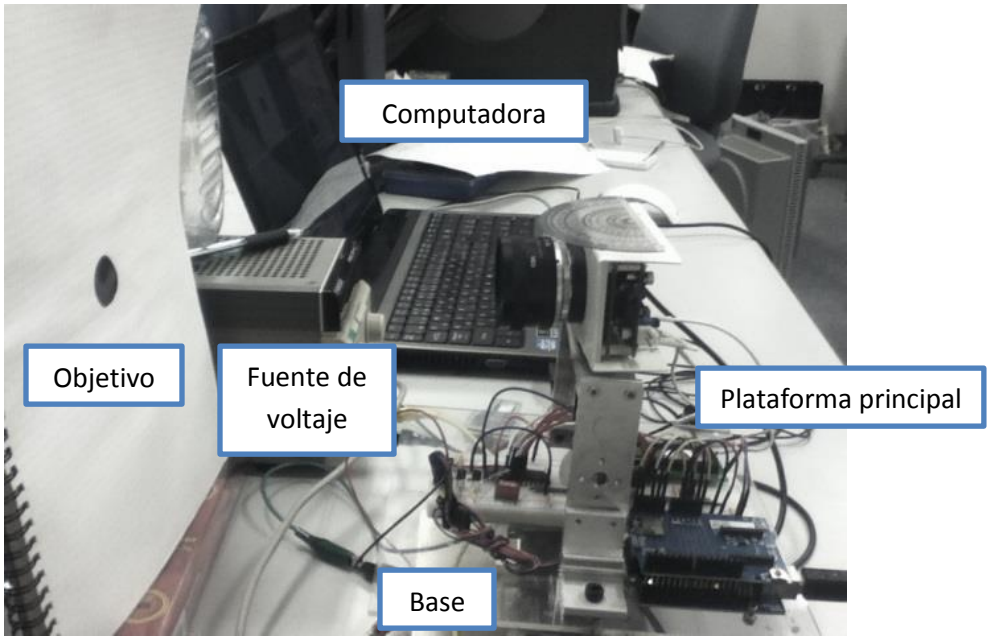


Fig.5.10: Fotografía de la plataforma

Por otro lado, en la Figura 5.11 se puede observar el procesamiento visual que la computadora realiza en tiempo real.

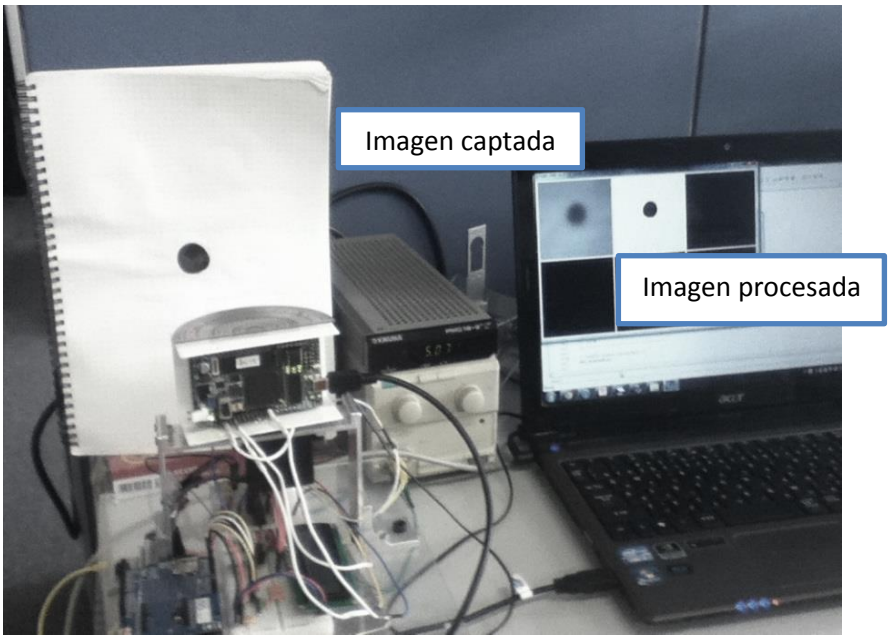


Fig. 5.11: Procesamiento de imagen hecho por computadora.

Las lecturas de los sensores pueden obervarse en la Figura 5.12. Nótese que el ángulo detectado por cada sensor es el mismo pero en dirección opuesta, debido a las propiedades de cada sensor, es decir, el giroscopio mide el ángulo de rotación directamente sobre la plataforma mientras que la cámara lo mide indirectamente al medir la distancia hacia el objetivo en la matriz de pixeles. También se puede observar una discrepancia en la amplitud y un offset en las dos gráficas. Esto se debe simplemente a que las escalas de los valores de los dos sensores no han sido igualadas en este punto.

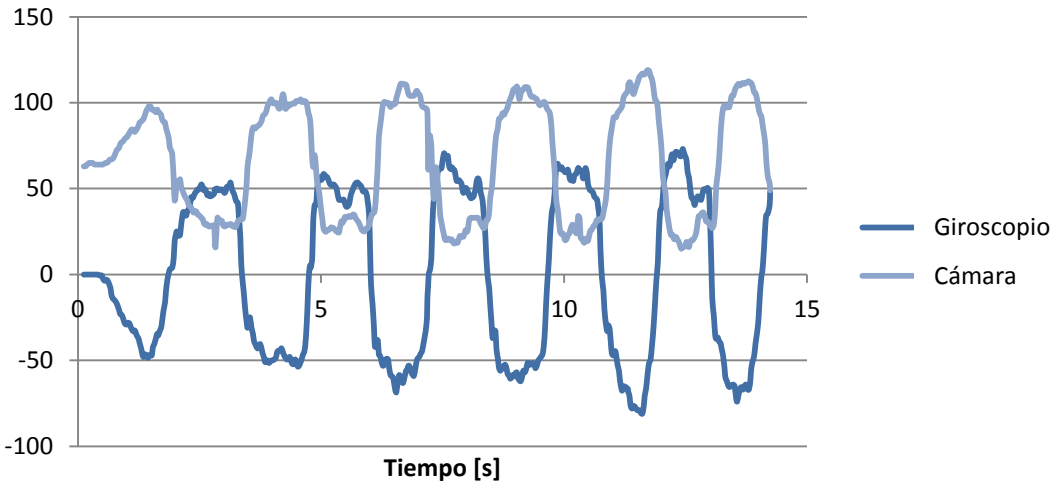


Fig. 5.12: Salida de la cámara y del giroscopio

Finalmente, en la Figura 5.13 puede observarse una comparación entre los datos de salida de la cámara y la combinación con los datos del giroscopio aplicando el algoritmo descrito. Obsérvese que la salida de la cámara tiene grandes saltos en ciertos puntos singulares. Se cree que esto se debe a un error en el

protocolo de comunicación entre el FPGA y el PSoC. Sin embargo, al combinar los datos con la información del giroscopio el error desaparece por completo y el sistema es capaz de desempeñarse sin problemas.

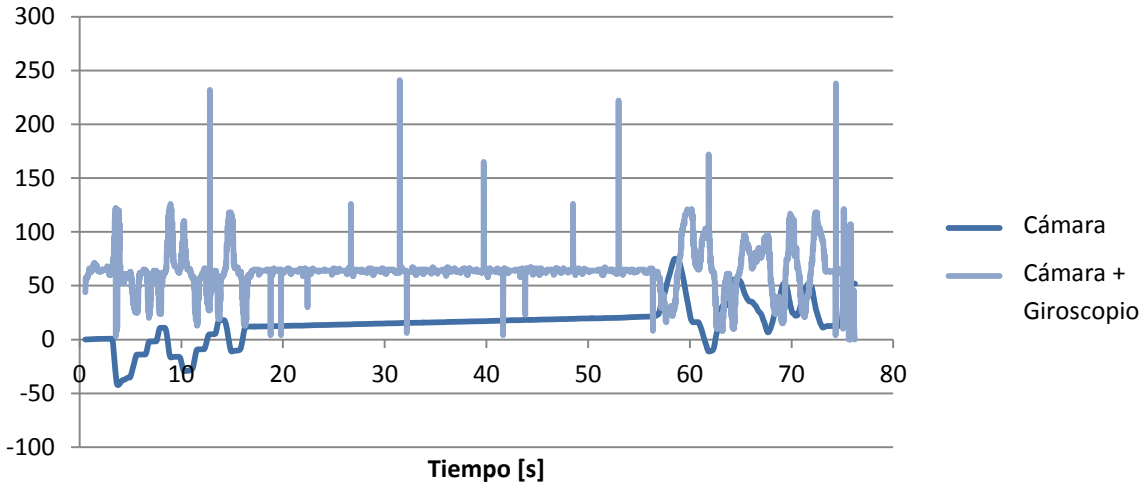


Fig. 5.13: Combinacion de valores.

Hasta este punto se logró comprobar de manera satisfactoria la combinación de los valores del giroscopio y la cámara en tiempo real experimentalmente, eliminando de manera satisfactoria los tiempos de retraso de la cámara y el sesgo en el valor de salida del giroscopio. En la siguiente sección se presentarán las conclusiones del proyecto y se presentarán algunas posibles mejoras a la plataforma como trabajo a futuro.

CONCLUSIONES Y TRABAJO A FUTURO

Los sistemas de estabilización visual han sido investigados y desarrollados desde hace aproximadamente 30 años, muchos investigadores han utilizado diferentes métodos y han cumplido satisfactoriamente con el objetivo principal. El presente documento presentó un método utilizando retroalimentación visual como la fuente de información principal acerca de la posición del sistema con el auxilio de un giroscopio para compensar por la baja frecuencia de muestreo del sensor visual.

Durante el desarrollo de este proyecto se probaron diferentes propuestas para resolver el problema planteado. Se probaron diferentes propuestas de hardware, utilizando diferentes sensores y protocolos para enfrentar el mismo problema. Se presentaron conceptos como los sistemas de control discreto y la transformada-z, los cuales fueron fundamentales en el análisis y desarrollo del algoritmo presentado.

El desempeño del sistema al implementar el algoritmo desarrollado fue satisfactorio, tanto en la simulación como en los resultados experimentales. El problema que presenta el sistema en su etapa actual es que su funcionalidad depende de ciertos parámetros que limitan su aplicación en la vida real.

Aunque el proyecto cumplió con los objetivos propuestos, las restricciones que presenta limitan su funcionamiento de tal forma que no puede considerarse para aplicaciones en el campo de trabajo. Una lista de mejoras y expansiones que pueden implementarse en el proyecto se presentan a continuación:

- ✓ **Aumentar los grados de libertad del sistema:** Actualmente, el proyecto fue diseñado para compensar el movimiento en un sola dirección angular, alrededor del eje z, es decir, de izquierda a derecha. La plataforma puede ser expandida para compensar también movimiento vertical (de arriba hacia abajo y viceversa), es decir, rotando alrededor del eje y. Con un código de procesamiento visual más complejo también podría cubrir rotación alrededor del eje x, esto es, rotación de lado a lado.

- ✓ **Integrar otros sensores inerciales:** La integración de otros sensores inerciales a la plataforma, por ejemplo, un acelerómetro, un magnetómetro o una antena GPS nos proveerían de mayor información acerca del ambiente y de la posición de nuestra plataforma en el mismo, así como de los agentes externos que pueden causar disturbaciones en el sistema (aceleración de la gravedad, aceleración del

objeto sobre el que va montado la plataforma, etc) lo cual puede aumentar la precisión del sistema de manera considerable.

- ✓ **Incrementar la robustez del sistema de procesamiento visual:** Una de las mejoras más importantes que pueden aumentar la practicidad y expandir las aplicaciones de la plataforma es integrarla con un sistema de detección de objetivos u otro tipo de procesamiento visual más avanzado. Ya que esta plataforma está diseñada para trabajar con sistemas móviles podría integrarse con un sistema de reconocimiento de caminos o de detección de obstáculos, teniendo así aplicación en sistemas de navegación autómata y en sistemas de seguridad.

- ✓ **Realizar más pruebas:** Para poder asegurar la confiabilidad del sistema en la industria o en aplicaciones de investigación se deben de realizar una serie de pruebas al sistema. Por ejemplo, hacer pruebas de movimiento lineal y observar y analizar el comportamiento de los sensores en este caso para determinar si se ven afectados de manera significativa.

REFERENCIAS

[1] E. Gamborino. *"Active Robotic Gaze Stabilization by Mechanical Compensation in 1 Degree of Freedom (DOF)"* BioSystems and Devices Laboratory, Graduate School of Engineering, Osaka University, Osaka, Japan. Agosto 2012.

[2] P. Corke, J. Lobo and J. Dias. *"An Introduction to Inertial and Visual Sensing",* CSIRO ICT Centre, Brisbane, Australia and University of Coimbra, Portugal. Junio 2007.

[3] T. Shibata and S. Schaal. *"Biometric Gaze Stabilization",* Kawato Dynamic Brain Project, ERATO, JST and USC.

[4] J. Soong, C. Brown. "*Inverse kinematics and gaze stabilization for the rochester robot head*", Rochester University, New York, 1991.

[5] T. Shibata, S. Schaal. *"Biomimetic gaze stabilization based on feedback-error-learning with nonparametric regression networks"*, Japan, United States, 2001.

[6] S. Takizawa. "*2DOF motion stabilization of biped robot by gaze control strategy*", Tohoku University, Japan, 2005.

[7] H. Sampieri. "*Metodología de la Investigación*", McGraw Hill, 2003.

[8] A. Luque. *"Diseño de un acelerómetro en PolyMUMPS (I)",*
Puede encontrarse en: http://iecon02.us.es/ASIGN/SEA/MEMS_TRABAJO1.pdf

[9] M. Kraft. *"Closed loop digital accelerometer employing oversampling conversion, Chapter 2. Principle of operation of an accelerometer"* Coventry University, School of Engineering, UK, 1997.

[10] *"±1.5g, ±6g Three Axis Low-g Micromachined Accelerometer", Freescale Semiconductor Inc,* Abril 2008.

[11] *"Understanding Single-Ended, Pseudo-Differential and Fully-Differential ADC Inputs", Maxim Integrated,* Junio 2002

[12] J. Borenstein. "*Systems and methods for robot mobile robot positioning*", Chapter 2: Heading Sensors. Puede encontrarse en: http://www.eng.yale.edu/ee-labs/morse/other/pos96ch2.pdf

[13] V. Apostolyuk. *"Theory and Design of Micromechanical Vibratory Gyroscopes"*, MEMS/NEMS Handbook, Springer, 2006

[14] *"ITG-3200 Product Specification, Revision 1.7"*, *InvenSense Inc,* Febrero 2011.

[15] T. Yagi, S. Kameda. "*An Analog VLSI Chip Emulating Sustained and Transient Response channels of the Vertebrate Retina",* IEEE transactions on neural networks, vol-14, no. 5, Septiembre 2003.

[16] *"FPGAs 1 CMPE691/491: Advanced FPGA Design"* University of Maryland Baltimore County, 2004.

[17] *"Xilinx FPGA products",* Puede encontrarse en: http://dsp-fpga.com/products/search/?q=xilinx+fpga&op=cn&max=40

[18] D. Smith. "*VHDL and Verilog Compared and Contrasted"* Puede encontrarse en: http://www.angelfire.com/in/rajesh52/verilogvhdl.html

[19] *"Introducción a la plataforma PSoC"* Puede encontrarse en: http://psoc-chile.es.tl/Home.htm

[20] "*CY8C29466 – PSoC™ Mixed Signal Array*" *Cypress MicroSystems, Inc,* Agosto 2004.

[21] D. Van Ess. *"AN2096 - PSoC™ 1, Using the ADCINC Analog to Digital Converter"*, Junio, 2011

[22] T. Dust. *"AN51234 - Getting Started with SPI on PSoC 1™"*, Noviembre 2011

[23] T. Dust. *"AN50987 - Getting Started with I^2C in PSoC™ 1"*, Noviembre, 2011

[24] D. Van Ess. *"AN2041 - PSoC™ 1, Understanding Switched Capacitor Filters"*, Septiembre 2011

[25] R. Klavon. *"ECE480 - UART Communication with the PSoC Development Kit"*, Marzo 2011

[26] *"Arduino Home Page*" Puede encontrarse en: http://www.arduino.cc

[27] *"LM150/LM350A/LM350 3-Amp Adjustable Regulators"*, *National Semiconductor Corporation,* Mayo 1998.

"

[28] Canny, J. "*A Computational Approach to Edge Detection"* IEEE Trans. Pattern Analysis and Machine Intelligence, 1986.

[29] *"OpenCV Blobs Library"* Puede encontrarse en: http://opencv.willowgarage.com/wiki/cvBlobsLib

[30] *"Serial Buses information page"* Puede encontrarse en: http://www.epanorama.net/links/serialbus.html

[31] D. Kalinsky, R. Kalinsky. *"Introduction to Serial Peripheral Interface"* Enero 2002. Puede encontrarse en:http://www.eetimes.com/discussion/beginner-s-corner/4023908/Introduction-to-Serial-Peripheral-Interface

[32] *"UM10204 – I2C Bus Specification and User Manual"* Puede encontrarse en: http://www.nxp.com/documents/user_manual/UM10204.pdf

[33] P. Albertos, A. Crespo. *"Real Time Control of Non Uniformly Sampled Systems",* Dept. of Systems Eng., Computers and Control, Universidad Politécnica de Valencia, Spain, 2009.

[34] F. Haugen. "*Discrete Time Signals and Systems*", TechTeach, Febrero, 2005.

[35] J. Sacerdoti. "*Transformada Z*", Departamento de Matemáticas, Facultad de Ingeniería, Universidad de Buenos Aires, 2003.

[36] V. Toochinda. "*Digital PID Controllers*", Junio 2011. Puede encontrarse en: http://www.controlsystemslab.com

[37] Starlino, *"A guide to using IMU (Accelerometer and Gyroscope Devices in Embedded Applications"*, Diciembre, 2009. Puede encontrarse en: http://www.starlino.com/imu_guide.html

[38] J. Borenstein, L. Ojeda, S. Kwanmuang. *"Heuristic Reduction of Gyro Drift in IMU-based Personnel Tracking Systems"*, The University of Michigan, USA, Abril 2009.

[39] G. Welch, G. Bishop. "*An Introduction to the Kalman Filter*", University of North Carolina at Chapel Hill, Chapel Hill, NC 27599-3175. Julio, 2006.

[40] D. Piyabongkarn, A. S. Sezen, R. Rajamani and B.J. Nelson, *"Development of a MEMS Gyroscope for Absolute Angle Measurement"*. Puede encontrarse en: http://www.me.umn.edu/labs/advcontrols/gyroscope.html

[41] "PSoC Family Graph", puede encontrarse en: http://www.cypress.com/psoc/images/Family%20Graph.jpg

APÉNDICE A

ARTÍCULO ORIGINAL REDACTADO EN YAGI BIOSYSTEMS, OSAKA UNIVERSITY

Robotic Active Gaze Stabilization by Mechanical Compensation in 1 Degree of Freedom (DOF).

Gamborino Castañeda Edwinn
Yagi Laboratory, BioSystems and Devices Area, Division of Electrical, Electronic and Information Engineering, Graduate School of Engineering, Osaka University, Osaka, Japan.
gamborino@neuron.eei.eng.osaka-u.ac.jp

***Abstract* – This paper gives a detailed introduction to the working principle of IMU sensors gyroscope and accelerometer, as well as to intelligent vision systems and a brief introduction to discrete control theory. All of this knowledge gets combined in order to solve the problem of gaze stabilization by mechanical means with sensors that have different sampling rates. A project was made out of this work and the results of the tests performed to said platform will be shown at the end of this document.**

I. INTRODUCTION

Active gaze stabilization is of vital importance for the use of high resolution tele-cameras in autonomous vehicles. Small aperture angles together with large focal lengths cause high sensitivity to rotational vehicle motion induced e.g. by bumps or braking. Due to large latencies in image processing gaze stabilization based only on image feedback is not fast enough to compute a real time response that is able to counteract said effects. Therefore, there is a necessity for a sensor with a faster response rate, such as inertial sensors, in particular angular rate sensors (gyroscopes) and linear acceleration sensors (accelerometers).

While there have been numerous attempts in recent years to provide active gaze stabilization through various methods, the approach taken by the following project was to be able to have a system with a low computational load that could be performed by embedded devices, therefore simplicity of design was always kept in mind for the design of this platform.

The following paper presents an experimentally tested algorithm to successfully combine the data provided from both sensors and the visual feedback, taking into account the sampling latencies of each individual sensor and the computing capabilities of embedded systems.

The information will be presented in the following way: in Section II a description of the inertial sensors is given, including their principle of operation, a description and communication protocols required by the specific integrated circuits used for experimental tests. Also in Section II a description of the visual sensor is given. The sensor used in the current experimentation is the Silicon Retina, developed by Tetsuya et al. [ref]. This sensor is controlled by an embedded FPGA Spartan III, which will also be briefly discussed in this section.

In Section III a description of the experimentation platform built is given, including mechanical, electric and electronic characteristics, and a detailed description on the transmission of information in between devices.

Section IV will describe the control diagram developed for the inertial sensors and the visual feedback in a separate and combined fashion. This description will include insight into the theory of discrete time systems and the programming interface of the platform.

Finally, the document will close with a description of the experiments performed, the results obtained from said experiments and we will draw some conclusions as well. The possibilities for future development of this project will also be discussed at the end of this section.

II. SENSORS DESCRIPTION

I. ACCELEROMETER

A. Introduction

Acceleration is a measure of how quickly speed changes. Just as a speedometer is a meter that measures speed, an accelerometer is a meter that measures acceleration. Accelerometers can be used to sense acceleration to measure a variety of things that are very useful to electronic and robotic projects and designs, such as: Acceleration, tilt and tilt angle, vibration, collision and gravity, among others.

B. Principle of Operation

1. Derivation of the Motion Equation

The measurement of acceleration always relies on classical Newton's mechanics. Normally the acceleration to which a body is subjected is of interest; the accelerometer being rigidly attached to that body. The transducers which are the used in the present paper can be modeled as a sensing element consisting of a proof mass (also referred to as seismic mass) which is suspended by a spring; acceleration causes a force to act on the mass which is consequently deflected by a distance 'x', as shown in Fig. 2.1. Some form of damping is required, otherwise the system would oscillate at its natural frequency 'f_n' for any input signal

Note that modern accelerometers do not use real physical suspended masses to measure acceleration, but rely on different techniques developed using MEMs.

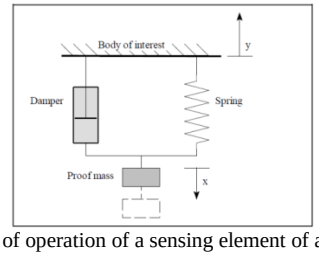


Fig. 2.1: Principle of operation of a sensing element of an accelerometer.

To derive the motion equation of the system, *D'Alembert's* principle is applied, where all real forces acting on the proof mass are equal to the inertia force on the proof mass. From the stationary observer's point of view, the sum of all forces in the y direction is:

$$m\frac{d^2(x(t)-y(t))}{dt^2}+\mathrm{b}\frac{dx(t)}{dt}+kx(t)=0$$

$$m\frac{d^2y(t)}{dt^2}=m\frac{d^2x(t)}{dt^2}+b\frac{dx(t)}{dt}+kx(t) \qquad [2.1]$$

Where:
m: mass of the proof mass,
y: movement of the body of interest,
x: movement of the proof mass,
b: damping factor,
k: spring constant.

From Equation [2.1] we can obtain the response for both steady and dynamic states as follows:

2. Steady state performance
In the steady state condition, that is with constant acceleration a, and constant deflection x, Equation [2.1] yields:

$$\mathrm{ma}=\mathrm{kx}$$

$$\mathrm{a}=\frac{\mathrm{kx}}{m} \qquad [2.2]$$

Here the sensitivity s of an accelerometer is defined by:

$$\mathrm{s}=\frac{x}{a}=\frac{m}{k} \qquad [2.3]$$

3. Dynamic state performance
For the dynamic performance it is easier to consider the Laplace transform of Equation [2.1]:

$$\frac{x(s)}{a(s)}=\frac{1}{s^2+\frac{b}{m}s+\frac{k}{m}}$$

From this equation, the natural frequency f_n is

$$f_n=\sqrt{\frac{k}{m}} \qquad [2.4]$$

While Equation [1.1] models the behavior of an accelerometer; even though parameters such as the sensitivity and the natural frequency are already given in the datasheet of any modern accelerometer, with equations [1.2] to [1.4], we could obtain all the necessary parameters to define our acceleration sensor.

C. **MMA7361L**

1. Description:
The MMA7361L is a low power, low profile capacitive micromachined accelerometer featuring signal conditioning, a 1-pole low pass filter, temperature compensation, self-test, 0g-Detect which detects linear freefall, and g-Select which allows for the selection between 2 sensitivities.
The device consists of a surface micromachined (See Fig. 2.2) capacitive sensing cell (g-cell) and a signal conditioning ASIC contained in a single package. The sensing element is sealed hermetically at the wafer level using a bulk micromachined cap wafer.

The g-cell is a mechanical structure formed from semiconductor materials (polysilicon) using semiconductor processes (masking and etching). It can be modeled as a set of beams attached to a movable central mass that move between fixed beams. The movable beams can be deflected from their rest position by subjecting the system to acceleration.

As the beams attached to the central mass move, the distance from them to the fixed beams on one side will increase by the same amount that the distance to the fixed beams on the other side decreases. The change in distance is a measure of acceleration.

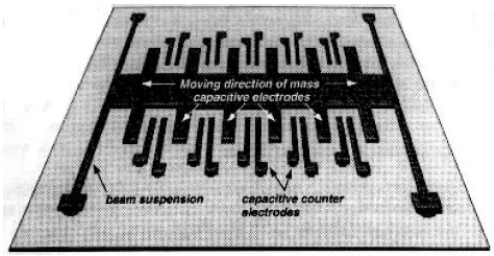


Fig. 2.2: Surface micromachined silicon accelerometer.

2. Electrical characteristics and pinout:

Parameter	Typical	Limit	Units
Supply voltage	3.3	2.2 to 3.6	V
Supply current	400	Up to 600	μA
Sleep mode current	3	10	μA
Operating temperature	-	-40 to 85	°C
Zero-g output	1.65	1.485 to 1.815	V
Sensitivity	800	740 to 860	mV/g
Sensitivity	206	190 to 221	mV/g
Output impedance	32	-	kΩ

Table 2.1: Operating characteristics of MMA7361L

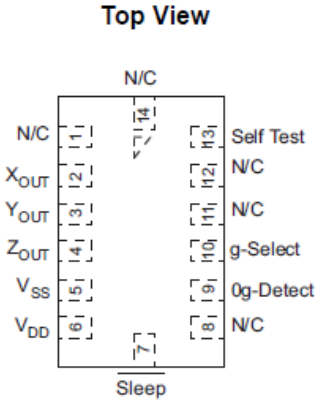


Fig. 2.3: Pinout description.

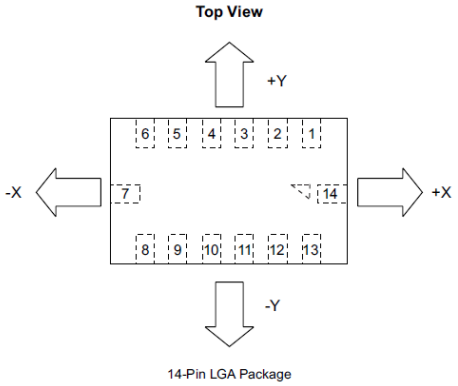


Fig. 2.4: Axis orientation of the MMA7361L.

D. ANALOG TO DIGITAL CONVERTER (ADC)

An Analog to Digital Converter (commonly known as ADC, A/D, among others) is an electronic device that converts an input analog (and continuous) voltage or current to a digital (and discrete) number proportional to the magnitude of the voltage or current.

For voltage input ADCs, three different input structure types exist: Single-Ended, Pseudo-Differential and Fully-Differential. Each type has its own trade-offs and the topology must be chosen accordingly to the kind of signal that needs to be discretized. A brief description of each topology is given in the following sections:

1) *Fully-Differential ADC:* Figure 2.5 shows an example of a fully-differential ADC T/H (Track and Hold) input structure. During track mode, $C_{sample(+)}$ charges to [$A_{IN(+)}$ Ð V_{DD}/2] and $C_{sample(-)}$ charges to [$A_{IN(-)}$ Ð V_{DD}/2]. When the T/H switches to hold mode, $Cs_{ample(+)}$ and $C_{sample(-)}$ connect together in series, such that the voltage sample presented to the ADC is the difference of $A_{IN(+)}$ and $A_{IN(-)}$. The differential architecture in conjunction with acceptable input bandwidth in the T/H are key ingredients for good dynamic common-mode rejection.

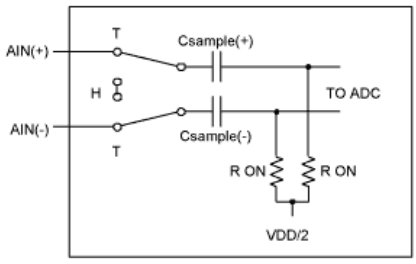


Fig. 2.5: Fully Differential T/H Stage.

2) *Pseudo-Differential ADC:* Pseudo-differential inputs are similar to fully-differential inputs in that they separate signal ground from the ADC ground, allowing DC common-mode voltages to be canceled (unlike single-ended inputs). However, unlike fully-differential inputs, they have little effect on dynamic common-mode noise. In Figure 2.6, sampling only occurs on the input $A_{IN(+)}$ signal. The signal common, $A_{IN(-)}$, is not sampled. During the 'Track' mode, the sampling capacitor charges through the series resistor R_{ON}. During the 'Hold' mode, the sampling capacitor connects to $A_{IN(-)}$ and an inverted input signal is presented to the ADC for conversion. Because sampling only occurs on the $A_{IN(+)}$ input, $A_{IN(-)}$ must remain within ±0.1LSB during the conversion for optimal performance.

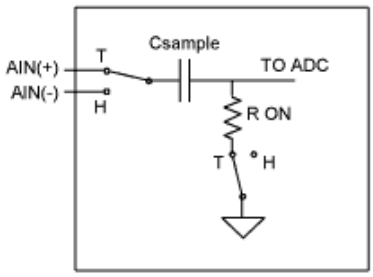


Fig. 2.6: Pseudo-Differential T/H Stage.

3) *Single-Ended ADC:* Single-ended inputs are generally sufficient for most applications. In single-ended applications, all signals are referenced to a common ground at the ADC. Each channel uses a single input pin. The analog ground pin is shared between all inputs for multi-channel systems. DC offset and/or noise in the signal path will decrease the dynamic range of the input signal. Single-ended inputs are ideal if the signal source and ADC are close to each other (i.e., on the same board so that signal traces can be kept as short as possible). Single-ended inputs are more susceptible to coupled-noise and DC offsets. However, signal conditioning circuitry can reduce these effects.

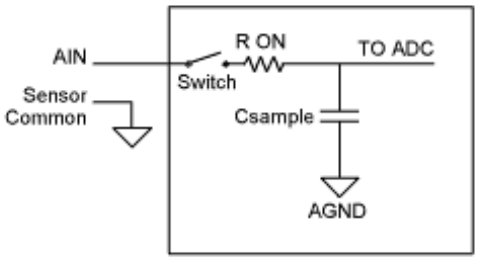


Fig. 2.7: Single-Ended T/H Stage.

Most microcontrollers with integrated ADC capabilities use the Single-Ended topology ADC, due to its simplicity of design and use.

E. ADC IN PSOC™ 1

The ADCINC is a differential or single input ADC that returns a 6 to 14 bit result. The maximum source clock frequency is 8 MHz, but 2 MHz is the maximum frequency recommended for improved linearity. This ADC may only be placed one time, due to its implementation which uses the hardware decimator rather than a digital block. This is the most resource efficient ADC. A 2nd order modulator may be implemented with an additional switch-capacitor block, allowing better linearity with an 8 MHz source clock.

Timing is implemented with an eight bit PWM that outputs a modulated pulse width that is synchronous to the input sample.
The ADCINC requires 2n–1 integration cycles to generate an output with n bits of resolution.

The ADCINC provides a first-order modulator formed from a single analog switched capacitor PSoC block, one digital PSoC block, and the decimator, as shown in the following figure:

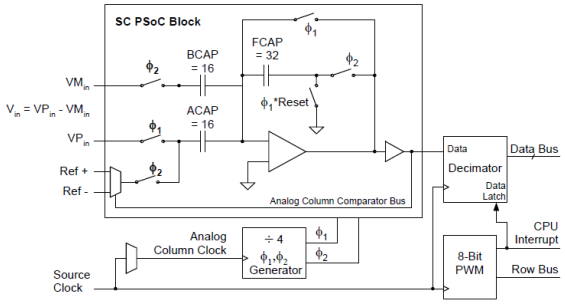


Fig. 2.8: Schematic of the ADCINC with First-Order Modulator.

The range of the ADCINC is set at $\pm V_{Ref}$, where V_{Ref} is set by the user in the Global Resources window of PSoC Designer. For fixed scale, V_{Ref} is set to $V_{Bandgap}$, or 1.6 $V_{Bandgap}$. For adjustable scale, V_{Ref} is set to Port 2[6]. For supply ratio metric scale, V_{Ref} is set to $V_{DD}/2$.

Table 1.2 below shows the DC and AC characteristics for the ADCINC module in PSoC1:

Parameter	Typical	Limit	Units
Input Voltage Range	-	Vss to Vdd	V
Input Capacitance	3	-	pF
Input Impedance	1/(C*clk)	-	Ω
Resolution	-	8 to 14	bits
Sample Rate	-	0.125 to 31.25	ksps
SNR	44	-	dB
DNL	0.6	-	LSB
INL	0.7	-	LSB
Offset Error	10	-	mV
Gain Error	3.0	-	%FSR
Operating Current	-	50 to 1900	μA
Data Clock	-	0.032 to 8	MHz

Table 2.2: 5.0V 1st-Order Modulator DC and AC Electrical Characteristics

II. GYROSCOPE

A. *Introduction*

A gyroscope is a device for measuring or maintaining orientation, based on the principles of angular momentum.

B. *Principle of Operation*

1. Mechanical gyroscope

Mechanically, a gyroscope is a spinning wheel or disk in which the axle is free to assume any orientation. Although this orientation does not remain fixed, it changes in response to an external torque much less and in a different direction than it would without the large angular momentum associated with the disk's high rate of spin and moment of inertia. Since external torque is minimized by mounting the device in gimbals, its orientation remains nearly fixed, regardless of any motion of the platform on which it is mounted.

The fundamental equation describing the behavior of the gyroscope is:

$$\tau = \frac{dL}{dt} = \frac{d(l\omega)}{dt} = \mathrm{I}\alpha \qquad [2.5]$$

where the pseudovectors τ and L are, respectively, the torque on the gyroscope and its angular momentum, the scalar I is its moment of inertia, the vector ω is its angular velocity, and the vector α is its angular acceleration.

It follows from this that a torque τ applied perpendicular to the axis of rotation, and therefore perpendicular to L, results in a rotation about an axis perpendicular to both τ and L. This motion is called precession. The angular velocity of precession ΩP is given by the cross product:

$$\tau = \Omega_p \, x \, L \qquad [2.6]$$

Under a constant torque of magnitude τ, the gyroscope's speed of precession ΩP is inversely proportional to L, the magnitude of its angular momentum:

$$\tau = \Omega_p L sin(\theta) \qquad [2.7]$$

where θ is the angle between the vectors ΩP and L. Thus, if the gyroscope's spin slows down (for example, due to friction), its angular momentum decreases and so the rate of precession increases. This continues until the device is unable to rotate fast enough to support its own weight, when it stops precessing and falls off its support, mostly because friction against precession cause another precession that goes to cause the fall.

By convention, these three vectors - torque, spin, and precession - are all oriented with respect to each other according to the right-hand rule.

While understanding the principle of operation of the Mechanical gyroscope, nowadays the gyroscopes that are found in most handheld electronic devices and used in many applications in the fields of robotics and aerospace engineering. The principle of operation behind this kind of gyroscope is explained in the following section.

2. Vibrating structure gyroscope

A vibrating structure gyroscope is a type of gyroscope that functions much like the halteres of an insect. The underlying physical principle is that a vibrating object tends to continue vibrating in the same plane as its support rotates. In the engineering literature, this type of device is also known as a Coriolis vibratory gyro because as the plane of oscillation is rotated, the response detected by the transducer results from the Coriolis term in its equations of motion.

Consider two proof masses vibrating in plane (as in the MEMS gyro) at frequencyω_r. Recall that the Coriolis effect induces an acceleration on the proof masses equal to $a_c = -2(vx\Omega)$, where v is a velocity and Ω is an angular rate of rotation. The in-plane velocity of the proof masses is given by: $x_{ip}\omega_r\cos(\omega_r t)$, if the in-plane position is given by $x_{ip}sin(\omega_r t)$. The out-of-plane motion y_{op}, induced by rotation, is given by:

$$y_{op} = \frac{F_c}{k_{op}} = \frac{2m\Omega x_{ip}\omega_r\cos(\omega_r t)}{k_{op}}$$

Where:
m is a mass of the proof mass,
k_{op} is a spring constant in the out of plane direction,
Ω is a magnitude of a rotation vector in the plane of and perpendicular to the driven proof mass motion.

These kind of electronic gyroscopes are packaged similarly to other integrated circuits and may provide either analog or digital outputs. In many cases, a single part includes gyroscopic sensors for multiple axes. Some parts incorporate both a gyroscope and an accelerometer, in which case the output has six full degrees of freedom.

Internally, MEMS gyroscopes use lithographically constructed versions of one or more of the mechanisms such as tuning forks, vibrating wheels, or resonant solids of various designs.

C. **ITG-3200A**

1. Description
The ITG-3200 features three 16-bit analog-to-digital converters (ADCs) for digitizing the gyro outputs, a user-selectable internal low-pass filter bandwidth, and a Fast-Mode I2C (400kHz) interface. Additional features include an embedded temperature sensor and a 2% accurate internal oscillator.

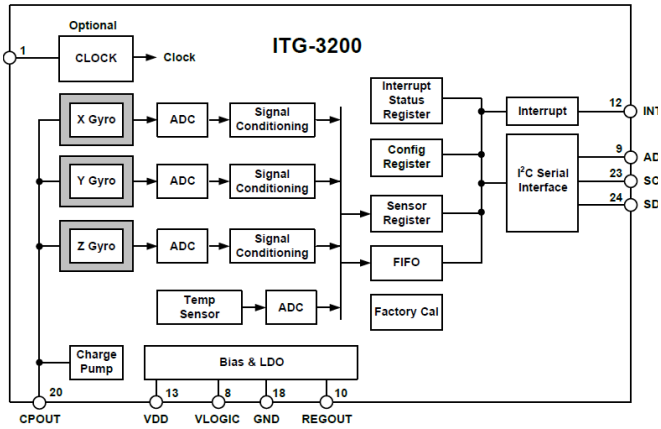


Fig. 2.9: ITG-3200 block diagram.

The ITG-3200 consists of three independent vibratory MEMS gyroscopes, which detect rotational rate about the X (roll), Y (pitch), and Z (yaw) axes. When the gyros are rotated about any of the sense axes, the Coriolis Effect causes a deflection that is detected by a capacitive pickoff. The resulting signal is amplified, demodulated, and filtered to produce a voltage that is proportional to the angular rate. This voltage is digitized using individual on-chip 16-bit Analog-to-Digital Converters (ADCs) to sample each axis.

The full-scale range of the gyro sensors is preset to ±2000 degrees per second (°/s). The ADC output rate is programmable up to a maximum of 8,000 samples per second down to 3.9 samples per second, and user-selectable low-pass filters enable a wide range of cut-off frequencies.

2. Electrical and physical characteristics

Parameter	Typical	Limit	Units
Supply voltage	2.5	2.1 to 3.6	V
Supply current	6.5	-	mA
Sleep mode current	5	-	μA
Operating temperature	-	-40 to 85	°C
Zero-g output	±40	-	deg/s
Full Scale range	±2000	-	deg/s
Sensitivity	14.375	-	lsb/deg/s
Sample rate	8	-	kHz

Table 2.3: Operating characteristics of ITG-3200A

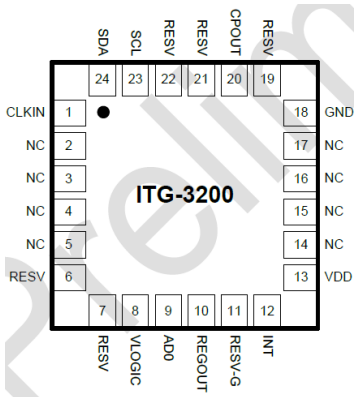


Fig. 2.10: ITG-3200 pinout.

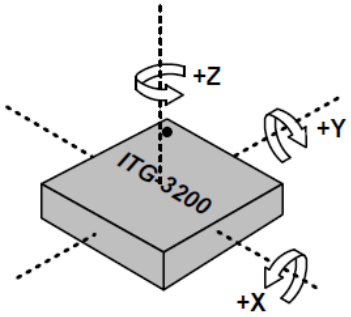


Fig. 2.11: Axial orientation on the ITG-3200.

3. Serial Communication Interface

The ITG-3200 communicates to a system processor using the I^2C serial interface, and the device always acts as a slave when communicating to the system processor. The slave address (AD) of the ITG-3200 devices is b110100X which is 7 bits long. The LSB bit of the 7 bit address is determined by the logic level on pin 9.

The ITG-3200 always operates as a slave device when communicating to the system processor, which thus acts as the

master. SDA and SCL lines typically need pull-up resistors to VDD. The maximum bus speed is 400kHz.

To write the internal ITG-3200 device registers, the master transmits the start condition (S), followed by the I2C address and the write bit (0). At the 9th clock cycle (when the clock is high), the ITG-3200 device acknowledges the transfer. Then the master puts the register address (RA) on the bus. After the ITG-3200 acknowledges the reception of the register address, the master puts the register data onto the bus. This is followed by the ACK signal, and data transfer may be concluded by the stop condition (P). This can be further understood in Figure 2.12:

MASTER	**S**	**AD+W**			**RA**			**DATA**		**P**
SLAVE			**ACK**			**ACK**			**ACK**	

Fig. 2.12: Single byte read sequence in I²C protocol with ITG-3200.

The I²C protocol will be thoroughly discussed in the next section.

D. INTER-INTEGRATED CIRCUIT (I²C) COMMUNICATION PROTOCOL

I²C is a two wire interface comprised of the signals serial data (SDA) and serial clock (SCL). In general, the lines are open-drain and bi-directional. In a generalized I²C interface implementation, attached devices can be a master or a slave. The master device puts the slave address on the bus, and the slave device with the matching address acknowledges the master. There are single master and multi-master modes included in the I²C specification; in this document we will only cover single master configuration, for more information on I²C protocol refer to [17].

A typical single master I²C configuration is depicted in Figure 2.13 below:

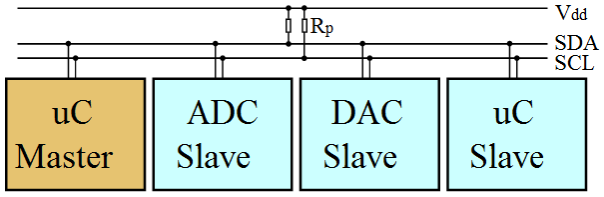


Fig. 2.13: A typical I²C single master configuration.

1. START (S) and STOP (P) Conditions

Communication on the I²C bus starts when the master puts the START condition (S) on the bus, which is defined as a HIGH-to-LOW transition of the SDA line while SCL line is HIGH. The bus is considered to be busy until the master puts a STOP condition (P) on the bus, which is defined as a LOW to HIGH transition on the SDA line while SCL is HIGH (see Figure 2.14 below).

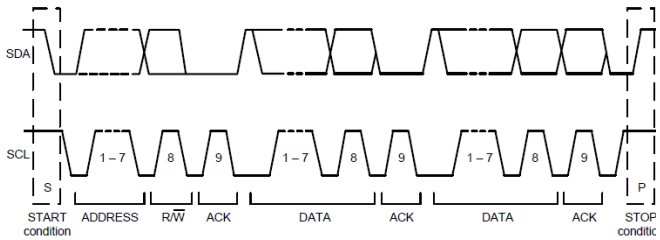


Fig. 2.14: Start and stop conditions on I²C bus.

2. Data Format / Acknowledge

I²C data bytes are defined to be 8 bits long. There is no restriction to the number of bytes transmitted per data transfer.

Each byte transferred must be followed by an acknowledge (ACK) signal. The clock for the acknowledge signal is generated by the master, while the receiver generates the actual acknowledge signal by pulling down SDA and holding it low during the HIGH portion of the acknowledge clock pulse.

If a slave is busy and cannot transmit or receive another byte of data until some other task has been performed, it can hold SCL LOW, thus forcing the master into a wait state. Normal data transfer resumes when the slave is ready, and releases the clock line (see Figure 2.15 below).

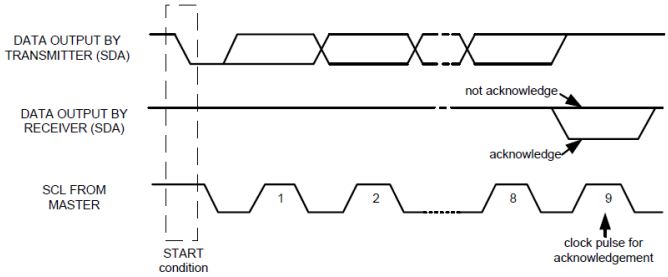


Figure 2.15: Acknowledge on the I²C bus.

3. Communications

After beginning communications with the START condition (S), the master sends a 7-bit slave address followed by an 8th bit, the read/write bit. The read/write bit indicates whether the master is receiving data from or is writing to the slave device. Then, the master releases the SDA line and waits for the acknowledge signal (ACK) from the slave device.

Each byte transferred must be followed by an acknowledge bit. To acknowledge, the slave device pulls the SDA line LOW and keeps it LOW for the high period of the SCL line. Data transmission is always terminated by the master with a STOP condition (P), thus freeing the communications line. However, the master can generate a repeated START condition (Sr), and address another slave without first generating a STOP condition (P). A LOW to HIGH transition on the SDA line while SCL is HIGH defines the stop condition. All SDA changes should take place when SCL is low, with the exception of start and stop conditions.

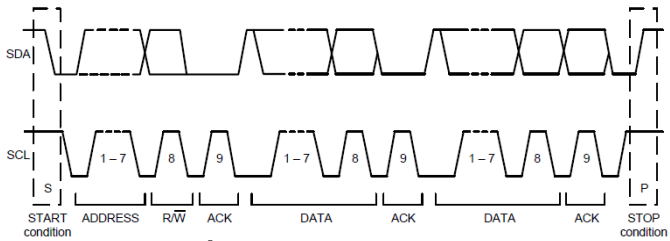


Figure 2.16: Complete I²C data transfer.

E. I²C IN PSOC™ 1

The I^2C user module provides support for an I^2C hardware resource. It is capable of transferring data at 50/100/400 kbps when the CPU clock is configured to run at 12 MHz. Faster or slower CPU clocks may be used, but may result in more or less bus stalling during address or data processing. The I^2C specification allows the master to run at clock speeds from 100 kHz down to DC. There are two different selections for SDA and SCL providing direct access to the hardware resource. Seven-bit address mode is supported in the supplied APIs. However, 10-bit addressing is supported with user extensions to the API set. This module does not require any analog or digital PSoC blocks to transfer data.

The I^2C resource supports data transfer at a byte-by-byte level. At the end of each address or data transmission/reception, status is reported or a dedicated interrupt may be triggered. Status reporting and interrupt generation is dependent on the direction of data transfer and the condition of the I2C bus as detected by the hardware. Interrupts may be configured to occur on byte-complete, bus-error detection and arbitration loss.

Every I^2C transaction consists of a Start, Address, R/W Direction, Data, and a Termination. The I^2C resource used for this User Module is capable of operating as either an I^2C Master or I^2C Slave. For either the Master or Slave operation, this User Module provides an interrupt based buffered transfer mechanism.

Communication is initiated from a foreground function call. At the completion of each byte of the message, an interrupt is triggered and the I^2C bus is stalled, the interrupt service routine (ISR) provided takes appropriate action on the bus allowing communication to continue, depending on the initialization performed by the user. A slave device which does not acknowledge an address will not be interrupted again until the next address is received. Slave devices must respond to each address either by acknowledging or not-acknowledging.

Ignoring differences between a Master and Slave, two general cases exist, that of a receiver and that of a transmitter. For an I^2C receiver, an interrupt will occur after the 8th bit of incoming data. At this point a receiving device must decide to acknowledge (ack) or not-acknowledge (nak) the incoming byte whether it is an address or data.

The receiving device then writes appropriate control bits to the I2C_SCR register, informing the I^2C resource of the ack/nak status. The write to the I2C_SCR register paces data flow on the bus by uninstalling the bus, placing the ack/nak status on the bus and shifting the next data byte in. For the second case of a transmitter, an interrupt will occur after an external receiving device has provided an ack or nak. The I2C_SCR may be read to determine the status of this bit. For a transmitter, data would be loaded into the I2C_DR register and the I2C_SCR register is again written to trigger the next portion of the transmission.

In addition to the buffer based transactions supported, the I^2C master User Module supports polled data transfers on a byte by byte basis without using the interrupt or supplied ISR.

Detailed descriptions of the I^2C bus and the implementation of the resource here may be obtained by referring to the complete I^2C specification available on the internet, and by referring to the device datasheet supplied with PSoC Designer.

III. SILICON RETINA

A. INTRODUCTION

The retina is a part of the central nervous system in the vertebrate and plays important roles in the early stage of visual information processing. Images projected on a two dimensional (2-D) photoreceptor array are transduced into electrical signals, and important information required for the brain to express higher order visual functions is extracted by the retinal circuit. The computations carried out in the retina are far more complex than those of simple CMOS imaging devices, as is suggested by the variety of light-induced response type of neurons. However, one can find two fundamental response types. One is a sustained-type response in which cells respond continuously during illuminations, and the other is a transient-type response in which cells respond transiently when illumination is turned on or off. The sustained response is thought to be relevant to the perception of static images and the transient response is thought to be relevant to the perception of moving objects.

Inspired by the unique architecture and algorithm of the retinal circuit, the neuromorphic silicon retinas, novel analog very large-scale integrated (VLSI) circuits, have been fabricated. Some silicon retinas emulate the center-surround receptive field of the outer retinal circuit. The silicon retinas mimicking the transient-type response were also fabricated to detect moving objects. Most of the chips fabricated so far, however, emulate either the sustained or the transient response exclusively, which limits applications of the silicon retina in real-time image processing. The silicon retina provides analog outputs mimicking the sustained and the transient responses.

B. Circuit description

The output of the chip emulating the sustained response has a Laplacian-Gaussian-like (∇^2G-like) receptive field and, therefore, carries out a smoothing and a contrast-enhancement on the input images. The output of the chip emulating the transient response is obtained by subtracting consecutive image frames that are smoothed in advance by a resistive network.

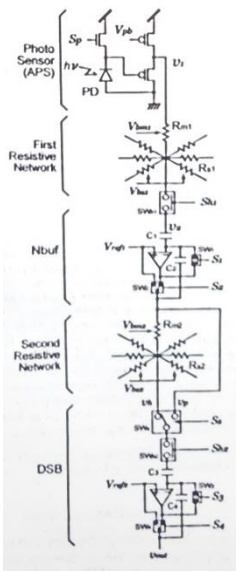


Fig. 2.16: Resistive network of a pixel in the silicon retina.

Fig. 2.16 shows the circuit design depicting a single pixel. The circuit was designed after the model of the vertebrate retina constructed with detailed physiological observations. The chip consists of two layers of resistive networks, which have different tightness of electrical couplings between neighboring pixels. A subtraction of outputs of these two resistive networks generates a ∇^2G-like receptive field. The photo sensor is an active pixel sensor (APS) which consists of a photo-diode and a source-follower circuit. The APS has a high sensitivity to light by accumulating photoelectrons in the parasitic capacitor of the photo-diode. Furthermore, the dynamic range can be controlled by changing the accumulation time.

Neighboring pixels are connected with MOS resistors at the first and second resistive networks. The resistances of the MOS resistors are controllable by external bias voltages. The NBUF and DSB are the sample/hold circuits.

Two fundamental types of image processing can be performed with the chip by controlling the external signals in different timings. The silicon retina was controlled by field programmable gate array (FPGA). A brief description of the advantages of the FPGA will be given in the following section.

Process	CMOS 0.8 um double-poly three-metal
Die size	8.9 x 8.9 [mm^2]
Number of pixels	127(H) x 127(V) [pixel]
Pixel area (PD area)	178.650 x 154.725 [um^2] 2:√3 (37.575 x 23.100 [um^2])
Fill factor	3.14 [%]
Power consumption (power saving off) (power saving on)	 367.9 mW (200.0 uW / pixel) @ 3.3 V 40.0 mW (21.7 uW / pixel) @ 3.3 V

Table 1.4: Specifications of the silicon retina

The chip had a power saving function to reduce power consumption. If the mode is active, current is not supplied to the Nbufs nor the DSBs, except for the DSBs in the row selected by the vertical shift register during a read out time. If the mode is inactive, current is always supplied to all the Nbufs and DSBs. The current version of the chip has 128x128 pixels. The chip was implemented with a 0.6 um, double-poly, three metal, standard analog CMOS technology and the die size was 8.9 x 8.9 mm^2. The power consumption without the power saving feature was 367.9 mW at 3.3 V power supply. The power consumption with the power saving function decreased to 40.0 mW. The specifications of the chip are presented in Table 1.

C. FPGA

The field-programmable gate array (FPGA) is a semiconductor device that can be programmed after manufacturing. Instead of being restricted to any predetermined hardware function, an FPGA allows you to program product features and functions, adapt to new standards, and reconfigure hardware for specific applications even after the product has been installed in the field—hence the name "field-programmable". You can use an FPGA to implement any logical function that an application-specific integrated circuit (ASIC) could perform, but the ability to update the functionality after shipping offers advantages for many applications.

Unlike previous generation FPGAs using I/Os with programmable logic and interconnects, today's FPGAs consist of various mixes of configurable embedded SRAM, high-speed transceivers, high-speed I/Os, logic blocks, and routing. Specifically, an FPGA contains programmable logic components called logic elements (LEs) and a hierarchy of reconfigurable interconnects that allow the LEs to be physically connected. You can configure LEs to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flipflops or more complete blocks of memory. One example of such devices is the Spartan-3 family, one of which is used to control the Silicon Retina.

The Spartan-3 family builds on the success of the earlier Spartan-IIE family by increasing the amount of logic resources, the capacity of internal RAM, the total number of I/Os, and the overall level of performance as well as by improving clock management functions. Numerous enhancements derive from the Virtex®-II platform technology.

These Spartan-3 FPGA enhancements, combined with advanced process technology, deliver more functionality and bandwidth per dollar than was previously possible, setting new standards in the programmable logic industry. Because of their exceptionally low cost, Spartan-3 FPGAs are ideally suited to a wide range of consumer electronics applications; including broadband access, home networking, display/projection and digital television equipment.

The Spartan-3 family is a superior alternative to mask programmed ASICs. FPGAs avoid the high initial cost, the lengthy development cycles, and the inherent inflexibility of conventional ASICs. Also, FPGA programmability permits design upgrades in the field with no hardware replacement necessary, an impossibility with ASICs.

While it is known that the Silicon Retina is controlled by the FPGA using DACs to change sampling rates among other control devices, for further reading about the interactions between the FPGA and the Silicon Retina one must refer to Yagi and Kameda [3]

III. Experimentation Platform

I. Introduction

While the broader scope of the project could include 2 or even up to 3 DOF (Degrees of Freedom), it was considered appropriate to start the simulation and experimentation with a simpler platform, this is with only one degree of freedom. Achieving full control over one dimension would assure us broader success when adding the rest without being so time consuming.

The selection of the axis was made purely by ease of experimentation, building the experimenting platform normal to the z-axis (pointing to the azimuth) proved to be the easiest to build and would convey the same information as any other axis, therefore axis selected was the x-axis; this is a rotation about the z-axis (known in flight dynamics as yaw rotation).

II. Main body

A. Mechanical design

Since the shape or size of the test platform was not a critical factor for the project, the platform was designed following the most basic restraints:

- ✓ Robust
- ✓ Lightweight
- ✓ Simple
- ✓ Inexpensive

In order to achieve these goals, the platform was built with recycled materials from other projects, keeping in mind the main functionality. Having to design only a one degree of freedom system was relatively simple and the manufacturing process was also fast and simple.

The only real restraints that were kept on mind at all times during the construction process were the interchangeability of parts and the interconnectivity amongst them. This is one of the reasons why a breadboard was kept until the end instead of a PCB layered card. Having and LCD was also necessary to be able to debug code and check errors as fast as possible.

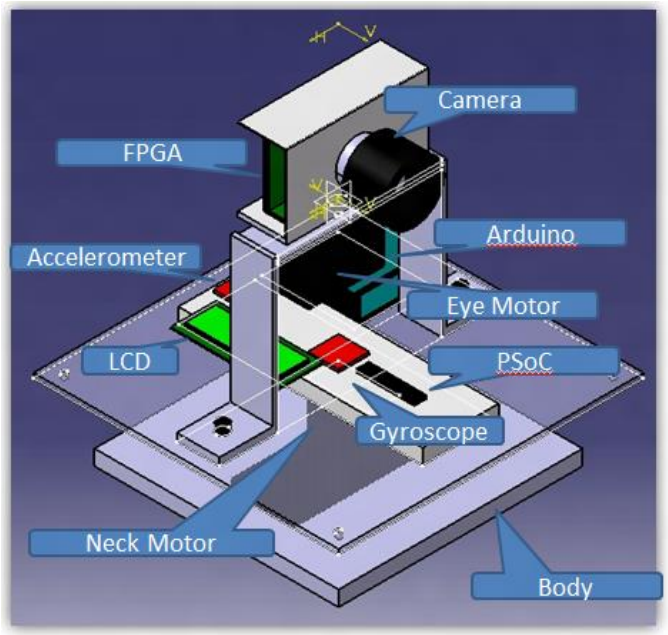


Fig. 3.1: The main body the system

B. Electric design

In terms of electric constraints, the system was designed in the beginning to work with 5 VDC, which would give the optimal output performance for the PSoC and Arduino. The first problem was that the operating voltage of the ITG-3200 is 3.3 volts. A voltage converter from 5 V to 3.3 V (Figure 3.2) was then designed to be able to feed the whole system with a single source.

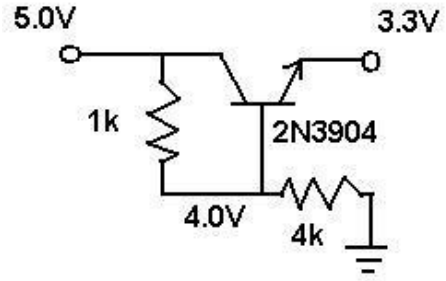


Figure 3.2: Voltage converter

The next problem found is that SPI communication wouldn't work properly between FPGA and PSoC if the latter was working at a different voltage than 3.3 V, thus the whole program and circuit was redesigned to be able to feed all PSoC, FPGA, Arduino and ITG3200 with a single source. The LCD module still needed to be connected to 5 VDC, so the source voltage would still be 5 V and the rest of the elements would be fed from the voltage converter.

Finally, when the PWM signal was sent from the PSoC to control the servo motor, the latter was unable to recognize the signal because the voltage was too low. A voltage converter for discrete signals was built using the following array of transistors (Figure 3.3).

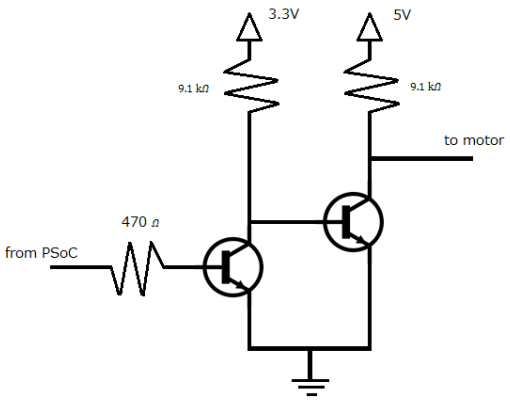


Fig. 3.3: Voltage translator from 3.3 to 5 V.

C. Electronic design

From an early stage in the project, it was determined that for a one axis rotation the information provided by the accelerometer is close to useless, so for the moment it was scraped from the original design.
On the electronic level, the main system consisted of three different CPUs, one FPGA and two microcontrollers (PSoC and Arduino), the visual sensor, the angular rate sensor, a servo motor, an LCD module and a computer for processing data.

The camera is controlled by the FPGA using a DAC converter and the feedback from the camera is concordantly conveyed via an ADC converter back to the FPGA. Even though this IC has the processing power to process the information by itself, in this stage of development, since the programming language it uses is VHDL, it is easier to send the information to the computer to be processed. Once the algorithm works then a program code can be hard written into the FPGA.

The FPGA is connected by SPI communication protocol to a PSoC 1 microcontroller (For a brief description of SPI protocol and its functionality in PSoC refer to paragraphs D and E of this section) in a one way only communication (this means that the slave sends data to the master all the time and the master doesn't need to acknowledge or send any signal to the slave). The FPGA works as the slave and the PSoC as the master. The only information that the FPGA is programmed to send is the x-axis position of the center of the black figure on a whit background in the screen. A brief description of how this is achieved is given next.

This is achieved by applying a double threshold to the image. By comparing every pixel, p(i,j) to its eight neighbors in brightness, we can set a range of values that will become white and a range of values that will become black.

Then the function to find the center of the black region is found by comparing all the pixels on each row and column with their corresponding column and row (for example, pixel p(1,1) would be compared with all the pixels from row one and column one) and find out, according to their relative position, where the center of the object is.

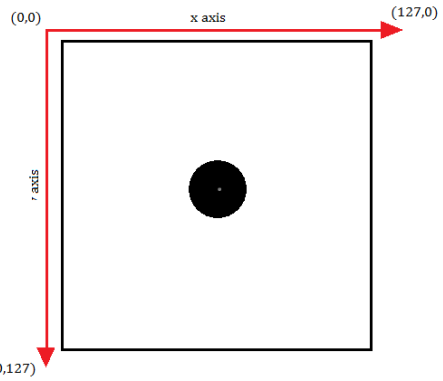


Fig. 3.4: Description of the image as seen by the visual sensor.

When the information is received by the PSoC microcontroller, it is processed to be turned into an angle that is fed into a PI closed feedback loop controller. The PI controller will be explained to further detail in Section IV.

At the same time, the digital gyroscope is gathering the information of the angular rate in all 3 axes, transforming this analog values to a 16 bit digital one and sending this information via I2C communication protocol (for more information about the inner workings of the gyroscope and the I2C protocol refer to Section II) to the PSoC microcontroller.

The PSoC then processes the information from both sensors, makes the necessary mathematic operations and performs the following operations simultaneously:

- Send relevant information to the LCD module for quick debugging
- Move the eye motor according to the result of the computations
- Send all the information via I2C protocol to the Arduino board to be logged into an SD card.

The Arduino then saves all the information to a CSV file in a Micro SD card, which can later be displayed on a computer for further analysis of the data acquired.

A scheme of the whole process including electric and electronic layers of interfacing can be observed in Figure 3.5 below.

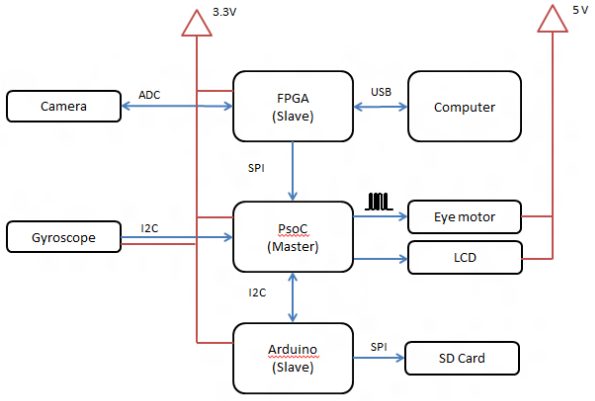


Fig. 3.5: Electric and electronic scheme of the main body.

D. SPI communication protocol

SPI is a serial bus standard established by Motorola and supported in silicon products from various manufacturers. SPI interfaces are available on popular communication processors such as the MPC8260 and microcontrollers such as the M68HC11. It is a synchronous serial data link that operates in full duplex (signals carrying data go in both directions simultaneously).

Devices communicate using a master/slave relationship, in which the master initiates the data frame. When the master generates a clock and selects a slave device, data may be transferred in either or both directions simultaneously. In fact, as far as SPI is concerned, data are always transferred in both directions. It is up to the master and slave devices to know whether a received byte is meaningful or not. So a device must discard the received byte in a "transmit only" frame or generate a dummy byte for a "receive only" frame.

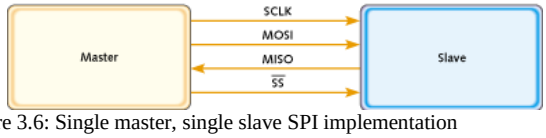


Figure 3.6: Single master, single slave SPI implementation

SPI specifies four signals: clock (SCLK); master data output, slave data input (MOSI); master data input, slave data output (MISO); and slave select (ÇSS). Figure 3.6 shows these signals in a single-slave configuration. SCLK is generated by the master and input to all slaves. MOSI carries data from master to slave. MISO carries data from slave back to master. A slave device is selected when the master asserts its ÇSS signal.

If multiple slave devices exist, the master generates a separate slave select signal for each slave. These relationships are illustrated in Figure 3.7.

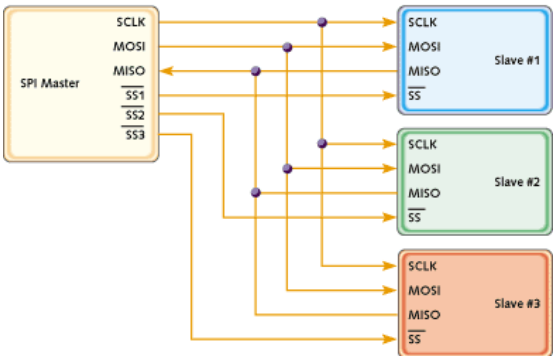


Figure 3.7: Single master, multiple slave SPI implementation

The master generates slave select signals using general-purpose discrete input/output pins or other logic. This consists of old-fashioned bit banging and can be pretty sensitive. You have to time it relative to the other signals and ensure, for example, that you don't toggle a select line in the middle of a frame.

While SPI doesn't describe a specific way to implement multi-master systems, some SPI devices support additional signals that make such implementations possible. However, it's complicated and usually unnecessary, so it's not often done.

A pair of parameters called clock polarity (CPOL) and clock phase (CPHA) determine the edges of the clock signal on which the data are driven and sampled. Each of the two parameters has two possible states, which allows for four possible combinations, all of which are incompatible with one another. So a master/slave pair must use the same parameter pair values to communicate. If multiple slaves are used that are fixed in different configurations, the master will have to reconfigure itself each time it needs to communicate with a different slave.

SPI does not have an acknowledgement mechanism to confirm receipt of data. In fact, without a communication protocol, the SPI master has no knowledge of whether a slave even exists. SPI also offers no flow control. If you need hardware flow control, you might need to do something outside of SPI.

Slaves can be thought of as input/output devices of the master. SPI does not specify a particular higher-level protocol for master-slave dialog. In some applications, a higher-level protocol is not needed and only raw data are exchanged. An example of this is an interface to a simple codec. In other applications, a higher-level protocol, such as a command-response protocol, may be necessary. Note that the master must initiate the frames for both its command and the slave's response.

SPI's full duplex communication capability and data rates (ranging up to several megabits per second) make it, in most cases, extremely simple and efficient for single master, single slave applications. On the other hand, it can be troublesome to implement for more than one slave, due to its lack of built-in addressing; and the complexity only grows as the number of slaves increases.

Far from being just a dumb "byte port," SPI is often an elegant solution for modest communication needs. It can also serve as a platform on which to create higher-level protocols.

E. SPI in PSoC™ 1

Features and Overview:

- ✓ Supports Serial Peripheral Interconnect (SPI) Master protocol
- ✓ Supports SPI clocking modes 0, 1, 2, and 3
- ✓ Selectable input sources for clock and MISO
- ✓ Selectable output routing for MOSI and SCLK
- ✓ Programmable interrupt on SPI done condition
- ✓ SPI Slave devices can be independently selected

The SPIM User Module is a Serial Peripheral Interconnect Master. It performs full duplex synchronous 8- bit data transfers. SCLK phase, SCLK polarity, and LSB First can be specified to accommodate most SPI clocking modes. Controlled by user-supplied software, the slave select signal can be configured to control one or more SPI Slave devices. The SPIM PSoC block has selectable routing for the input and output signals, and programmable interrupt-driven control.

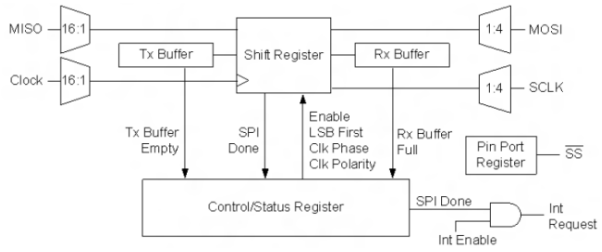


Fig.3.7: SPIM diagram.

SPIM is a user module that implements a Serial Peripheral Interconnect Master. It uses the Tx Buffer, Rx Buffer, Control, and Shift registers of a Digital Communications Type PSoC block and one or more Pin Port registers.

The Control register is initialized and configured using the Device Editor and/or the SPIM User Module firmware Application Programming Interface (API) routines. Initialization includes setting the LSB First configuration and the SPI transmission/receives clocking modes. SPI modes 0, 1, 2, and 3 are supported. Both the SPI master and slaves must be set with the same clock mode and bit configuration in order to properly communicate. The SPI modes are defined as listed in Table 1:

Mode	SCLK Edge performing data latch	Clock polarity	Notes
0	Leading	Inverted	Leading edge latches data, Data changes on trailing edge
1	Leading	Non Inverted	
2	Trailing	Inverted	Trailing edge latches data, Data changes on leading edge
3	Trailing	Non Inverted	

Table 2. SPI Modes

The active low slave select signal(s), ~SS, must be set with user-supplied software routines to control selected Pin Port register bits, enabling the SPI Slave devices properly. One or more slave select signals can be configured, but only one slave select signal can be active at a time.

The SCLK signal is the SPI transmit/receive clock. It is one-half the clock rate of the input clock signal. The effective transmit/receive bit rate is the input clock divided by two. The input clock is specified using the Device Editor. The MOSI signal is the Master-Out-Slave-In data signal that transmits the data from the master to a slave SPI protocol-compliant device. The MISO signal is the Master-In-Slave-Out data signal that transmits the data from the slave SPI device to this user module.

The SPIM hardware transmits data from the master SPI device on the MOSI signal and simultaneously receives data from the selected slave SPI device on the MISO signal. The same SCLK signal is used for both transmit and receive of the master and slave data.

III. EXPERIMENTAL BASE

Due to the necessity to obtain quantitative and reliable data from experiments performed repeatedly, an experimental base was devised.

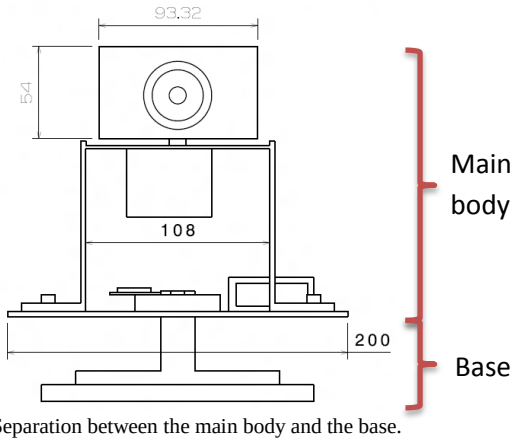


Fig. 3.8: Separation between the main body and the base.

The only function of this base would be to rotate the main body in a range from -90 to 90 degrees, where zero is the camera pointing at the black dot in the white background straight. Other parameters to consider would be to change the speed of rotation and the limits of the rotation (for example 0 to 60 degrees at 1 Hz or -30 to 30 degrees at 0.5 Hz), this values were enabled to be changed by analog controllers such as buttons and potentioneters. A electric and electronic diagram of the system can be observed in Figure 3.9.

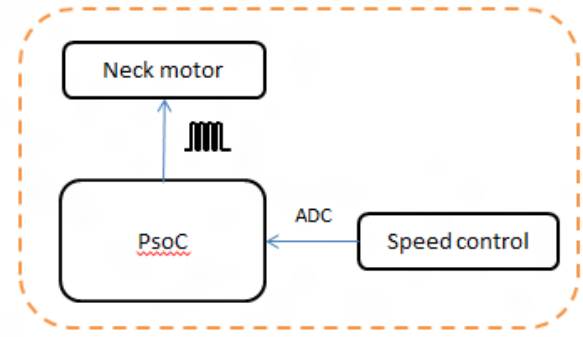


Fig. 3.9. Electric and electronic scheme of the base

IV. CONTROL DIAGRAM

I. INTRODUCTION

A. Description of the system

The objective of the current platform is to make the camera (eye) follow the target by rotating in the opposite direction of which the platform (head) is moving while keeping a reference to the base (ground) .

This system can be observed in Figure 4.1. The angles that we have to keep in mind are:

- ✓ Θh/g: Angle between head and ground, measured by gyro.
- ✓ Θt/e: Angle between target and eye, measured by camera
- ✓ Θe/h: Angle between eye and head, output fed to the motor

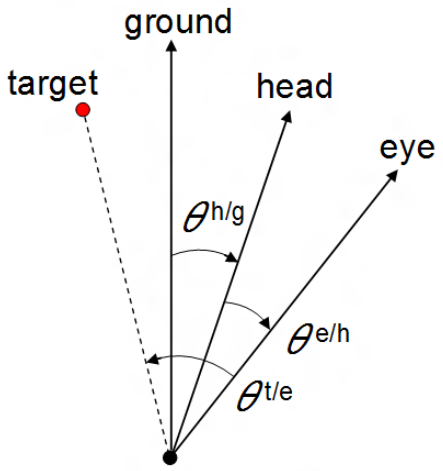


Fig. 4.1: Angle configuration for the platform

These relationships might be better observed on Figure 4.2. The red circles represent the three different movable segments of the platform and the blue arrows represent the angle vectors of each part, which, in the standard position, are all aligned to the front facing the target.

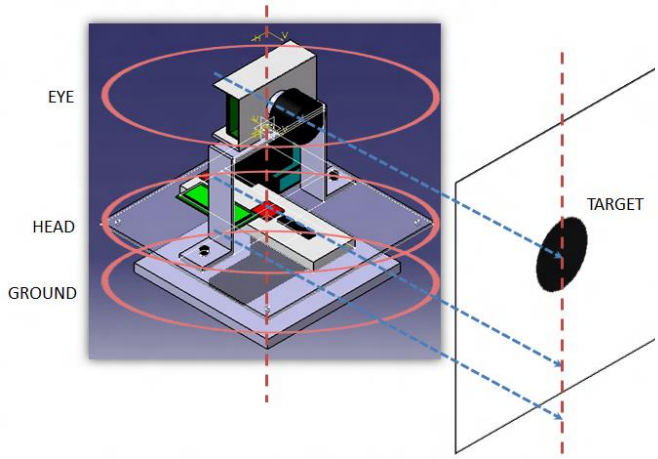


Fig. 4.2: 3D model depicting the different rotating parts and their position vectors as presented in Figure 4.1.

B. Discrete sampling control diagrams

In classical digital control system design, all the sampled data signals are assumed to be regularly, synchronously, and equally time spaced, being T the sampling period, but this is not the case in most industrial applications.

There are many reasons to force a different sampling scheme. For instance, the control signal can be updated at a fixed rate but the output vector components are measured by different sensors, each one having a maybe different sampling rate, noise characteristics and reliability. In some practical cases the output is not available at every sampling time. Another interesting and common situation is the distributed control where multiloop control systems, in which sampling rates are not the same and even not synchronised, are cooperating. This situation also raises the use of multirate control schemes, where the selection of the sampling rate is based on its suitability to control each controlled variable. This imposes the use of discrete time (DT) models with different sampling period, even for the same process.

A control system which uses a digital computer as a controller or compensator is known as digital control system. The advantages of using digital computers for compensation include: accuracy, reliability, economy and most importantly, flexibility. A block diagram of a digital control system is shown in the figure:

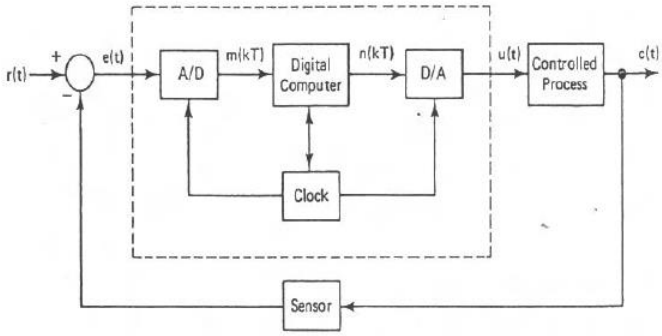


Fig. 4.3: Digital control system

A sampler is a device that converts an analog signal to a digital signal (train of pulses) while a hold device converts digital signal to analog signal by holding the value of the input pulse for a prescribed time duration. A sample-and-hold device performs both functions of sampling and holding. This process can be observed in Figure 4.4:

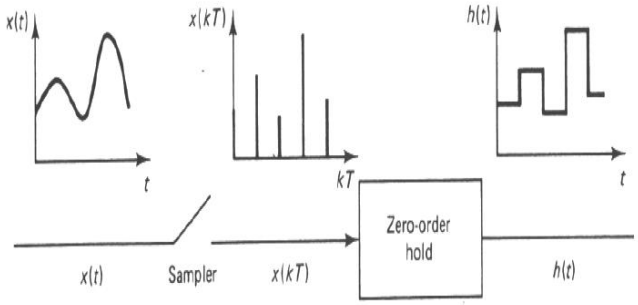


Fig. 4.4: Conversion from a continuous time signal to a discrete time signal in a control diagram.

These concepts will be used to explain the discrete closed loop controller that was developed for the current platform.

II. GYROSCOPE CONTROL

Since the output of the gyroscope is fed to the motor and, if we consider only the gyroscope part of the control diagram, there is no feedback from the position of the motor to check the error, so we can consider this part of the control to be an open loop controller.

Using the concepts explained earlier about discrete control systems and adding a proportional constant to adjust the values of the output while integrating it because we require knowing the angular position of the system, we end with the diagram shown in Figure 4.5:

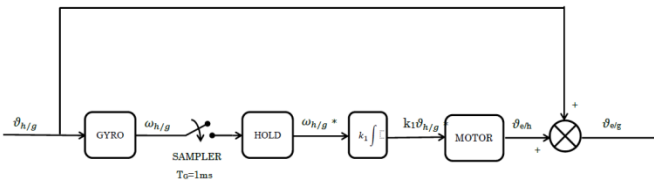


Fig. 4.5: Control diagram for the gyroscope

III. CAMERA CONTROL

A. *Discrete PID controller.*

Proportional-Integral-Derivative (PID) control is still widely used in industries because of its simplicity. No need for a plant model. No design to be performed. The user just installs a controller and adjusts 3 gains to get the best achievable performance. Most PID controllers nowadays are digital.

A standard "textbook" equation of PID controller is

$$\mathrm{u(t)} = \mathrm{K(e(t)} + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{dt}) \qquad [4.1]$$

where the error e(t), the difference between command and plant output, is the controller input, and the control variable u(t) is the controller output. The 3 parameters are K (the proportional gain), T_i (integral time), and T_d (derivative time).

Performing Laplace transform on (1), we get:

$$\mathrm{G(s)} = \mathrm{K}(1 + \frac{1}{sT_i} + sT_d) \qquad [4.2]$$

Another form of PID that will be discussed further in this document is sometimes called a parallel form.

$$\mathrm{u(t)} = \mathrm{K}_p\mathrm{e(t)} + K_i\int_0^t e(\tau)d\tau + K_d\frac{de(t)}{dt}) \qquad [4.3]$$

With its Laplace transform

$$\mathrm{G(s)} = K_p + \frac{K_i}{s} + \mathrm{s}K_d \qquad [4.4]$$

We can easily convert the parameters from one form to another by noting that

$$\begin{aligned} K_p &= K \\ K_i &= \frac{K}{T_i} \\ K_d &= KT_d \end{aligned} \qquad [4.5]$$

For digital implementation though, we are more interested in a Z-transform of (3)

$$\mathrm{U(z)} = \left[K_p + \frac{K_i}{1-z^{-1}} + K_d(1-z^{-1})\right]E(z) \qquad [4.6]$$

Rearranging gives

$$\mathrm{U(z)} = \left[\frac{(K_p + K_i + K_d) + (-K_p - 2K_d)z^{-1} + K_d z^{-2}}{1-z^{-1}}\right]E(z) \qquad [4.7]$$

Define:

$$\begin{aligned} K_1 &= K_p + K_i + K_d \\ K_2 &= -K_p - 2K_d \\ K_3 &= K_d \end{aligned} \qquad [4.8]$$

(7) can then be rewritten as

$$\mathrm{U(z)} - z^{-1}\mathrm{U(z)} = [K_1 + K_2 z^{-1} + K_3 z^{-2}]E(z) \qquad [4.9]$$

Which then converted back to difference equation as:

$$\mathrm{u(k)} = \mathrm{u(k-1)} + K_1 e(k) + K_2 e(k-1) + K_3 e(k-2) \qquad [4.10]$$

Which is suitable for implementation in our embedded device. The implementation of this formula in our algorithm will be further discussed in the next section.

B. *Implementation*

The camera is controlled by a simple feedback closed loop controller. Using the discretizing blocks and the PI controller (the derivative part has not been implemented so far) as explained in the previous sections, the control diagram can be drawn as follows:

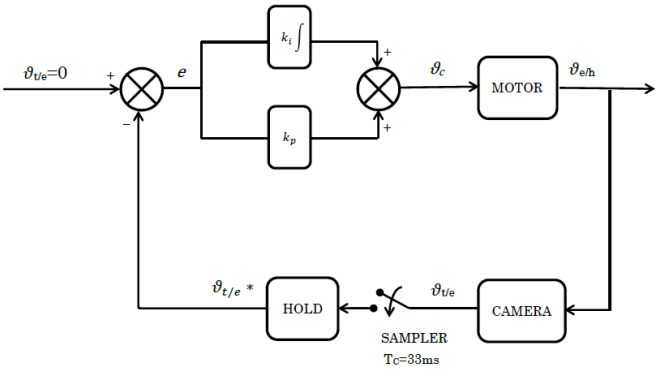


Fig. 4.6: Control diagram for the camera

IV. COMBINED CONTROL

The complete control diagram of the system is pretty much a combination of the two separated, with the addition of an element to describe the phenomenon that occurs when the program selects the data from the camera or adds the current data from the gyro to the previous data from the camera.

The period of the sampler of the camera is of 33 [ms] while the sampler of the gyro works at 2 [ms], therefore, for the remaining 32 milliseconds when we don't have updated data available from the camera the value of the angle according to the gyro gets held and the difference for each millisecond then is calculated and added to the last value obtained from the camera, therefore successfully combining both values and feeding the result to the motor. The performance of this algorithm is explained in detail in Section V.

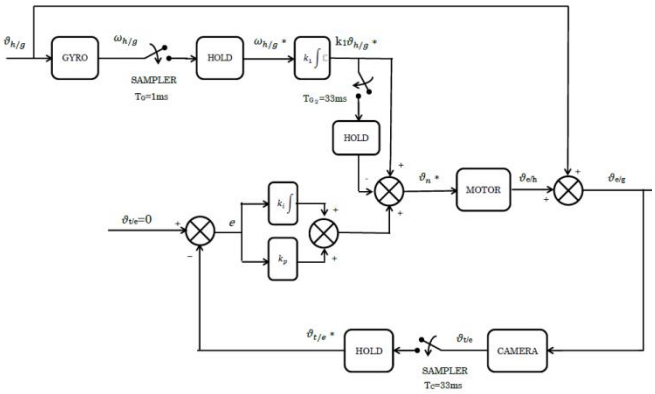


Fig. 4.7: Complete control diagram for the whole system

V. ALGORITHM

In this section the programming algorithms to get the angle from our sources and the way they were combined is thoroughly explained.

A. Angle according to the gyroscope

The following algorithm is based on some ideas from the Kalman filter[3][12], namely the comparison between the previous and the next state of the system to achieve a cleaner output, but it does not require heavy math, such as matrix algebra, therefore simplifying the burden for the CPU of our embedded system making the system faster.
The data from the gyro is a originally an discretized analog value that is directly transformed to a 16-bit value inside the gyro and presented to the microcontroller as digital data already. This data is obtained, as previously explained, via the via the I^2C communication protocol for the digital gyroscope. This value need to be translated into its real life counterpart.

These values are a 16-bit discretized representation, proportional to the voltages generated in the sensor. In order to match each discrete value to its corresponding voltage the following equation is used:

$$a_v = \frac{a_{16} v_{ref}}{step_{16}} \quad [4.11]$$

Where a_{16} is the discrete 16-bit value, v_{ref} is the voltage reference of the ADC module and step is the number of steps in a 16-bit value A 16 bit number has 2^{16} = 65,536 steps of accuracy. In the case of an analog sensor, the converted discrete voltage value may be proved by measuring the output voltage and comparing it with the value obtained from the ADC.

Next, to transform the voltage to the units the sensors measure (namely g (m/s^2) for the accelerometer and deg/s for the gyroscope) we may apply the following equation:

$$a_x = \frac{a_v - v_{zero}}{sensitivity} \quad [4.12]$$

Where v_{zero} is the voltage corresponding to a zero measurement for a given axis in the sensor at a given time and the *sensitivity*, usually expressed in mV/unit, is a constant expressing the proportionality between voltage and the actual units the sensor is supposed to measure. Both values can be found in the respective datasheet of each sensor.

Combining Equations 3.1 and 3.2 we get the full conversion equation from 16-bit discrete value to g and deg/s for the accelerometer and the gyroscope, respectively:

$$a_x = \frac{\frac{a_{16} v_{ref}}{step_{16}} - v_{zero}}{Sensitivity} \quad [4.13]$$

Once all of these operations are performed we have the value for the angle rate of any axis, though for the purposes of this project we will only use the information from the rotation about the z-axis. In the continuous time plane we could obtain the position from the velocity integrating over time:

$$\theta = \int_0^t \omega d\tau \quad [4.14]$$

But since we are working with discrete values we need to approach this problem using difference equations (not differential equations), knowing that the velocity is the change of position over time we can write:

$$\omega(i) = \frac{\theta(i) - \theta(i-1)}{\Delta t} \quad [4.15]$$

So isolating our interest variable ϑ(i) from Equation [4.15], we get:

$$\theta(i) = \omega(i)\Delta t + \theta(i-1) \quad [4.16]$$

Instead of using only the current value of ω(i) we calculate an average between the previous and the current reading:

$$\omega_{est} = \frac{\omega(i) + \omega(i-1)}{2} \quad [4.17]$$

The behavior of this calculation can be observed on Figure 4.8:

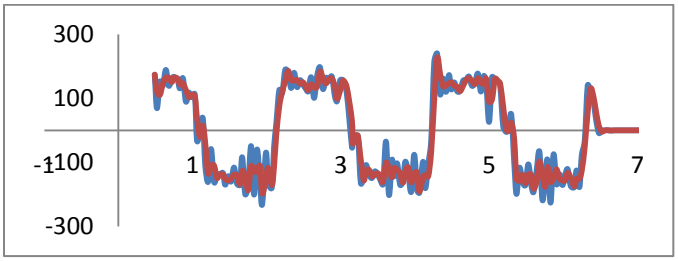


Figure 4.8: Noise reduction using an average of the current sample and the last sample (in red)

Finally substituting Eq. 4.17 on Eq. 4.16 we get:

$$\theta(i) = \omega_{est}\Delta t + \theta(i-1) \quad [4.18]$$

The results of this algorithm can be observed in Figure 4.8. We can see that the algorithm manages to send a very clean output of the angular position of the camera despite the noisy input. This

output only has the problem that the output value drifts with time. This phenomenon will be discussed in more detail in Section V.

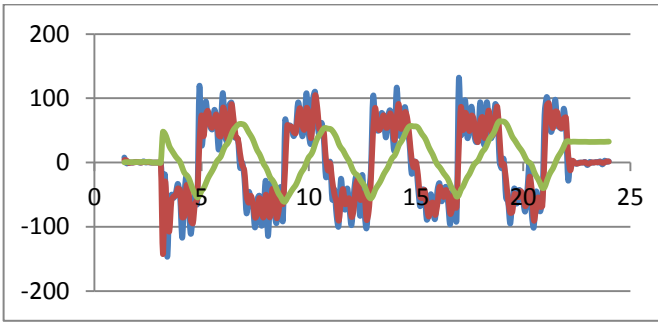


Figure 4.8. Original input (blue) averaged input (red) angle output (green)

B. Angle according to the camera

As explained in the previous section, the output of the camera was put under a PID controller to achieve a better performance. To better explain the process of getting the values for the camera a piece of the code programmed for the PSoC is presented:

```
/* PID controller for camera */
cRXData_R = cRXData;
u_last = u;

DV = 0x40;
kp = 0.6;
ki = 0.4;

k1 = kp+ki;
k2 = -kp;

e1 = e;
e = DV-cRXData_R;
delta_u = k1*e + k2*e1;
u = u + delta_u;
if(u<=750){u=750;}
if(u>=2500){u=2500;}
```

cRXData is the value of the x-axis coordinate of the center of the target as it is received from the FPGA. This value is saved into a new variable, cRXData_R to be manipulated by the PID controller.

The first step is to update the value used as the last outputted value for the algorithm, and then the constants according to Equation [4.8] are defined and finally, according to Equation [4.10] delta_u are calculated.

The input is updated and the program restarts.

The last part of the code is just a measure to prevent the counter to overflow causing a glitch in the program.

C. Combination of values

The values obtained from the previous calculations were combined using a very simple piece of code:

```
/* Combining values */
if (u_last-u != 0)
{
output = u;
}
else
{
output = u_last + delta_theta_m;
}

PWM16_1_WritePulseWidth(output);
```

Where u and u_last are the current and previous values according to the PID controller used in for the camera respectively, and delta_theta_m is the output value according to the camera.

The effect of this piece of code can be read as the following sentence: “If the value sampled by the camera has been updated, use that value only to feed the motor, if it hasn’t been updated, sum the current value of the gyro plus the previous angle of the camera”. The results of applying this code to the algorithm will be shown and discussed in the following sections.

V. RESULTS

1. DRIFT TESTS

Gyros and accelerometers have an offset, known as bias. The gyro bias for instance, shows itself after integration as an angular drift, increasing rise linearly over time (ramp). To compensate for sensor drift, a process called augmentation is used, whereby gyro and acceleration errors are compensated by utilising other system states. For instance the angular drift error of a gyro after integration can be compensated, if true angular measurements are taken. Comparing the two shows then the gyro drift. In our case, the drift of the gyroscope is compensated by the angle from the camera. In the following section, the complete results obtained from the tests performed to the gyroscope to determine its drift are presented. Several tests were performed according to the following algorithm:

1. Set the camera at a 30 degree counterclockwise angle from its initial position.

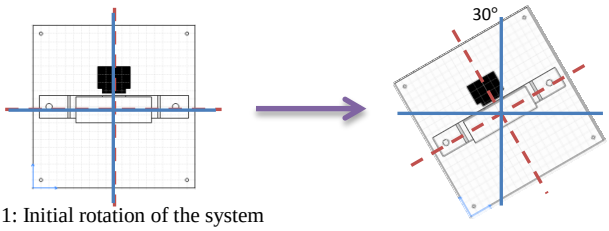


Fig. 5.1: Initial rotation of the system

2. Make the whole head turn 60 degrees clockwise, and then 60 degrees counterclockwise until reaching the initial position.

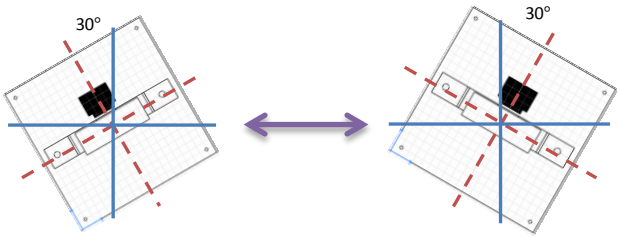


Fig. 5.2 Cyclic rotations of the system

3. Repeat the process until the established number of cycles is reached.

The same process then will be repeated several times for each number of cycles to obtain a qualitative value for the drift of the gyroscope. The results obtained are as follows:

TEST 1: 10 cycles.
Time average = 24.18 [s]. Angle displacement average = 13.2 [deg]. Drift average = 0.5459 [deg/s]

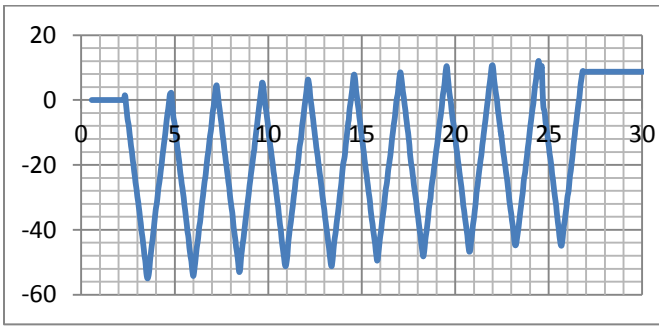


Figure 5.3: Random sample from Test 1.

Source	time [s]	total drift [deg]	drift speed [deg/s]
logger	24.5	10	0.408163265
	23	15	0.652173913
	24	13	0.541666667
	24	18	0.75
	24.5	14	0.571428571
	24.18	13.2	0.545905707

Table 5.1: Output results for Test 1.

Test 2. 20 cycles.
Time average = 49.1 [s]. Angle displacement average = 22 [deg]. Drift average = 0.4483 [deg/s]

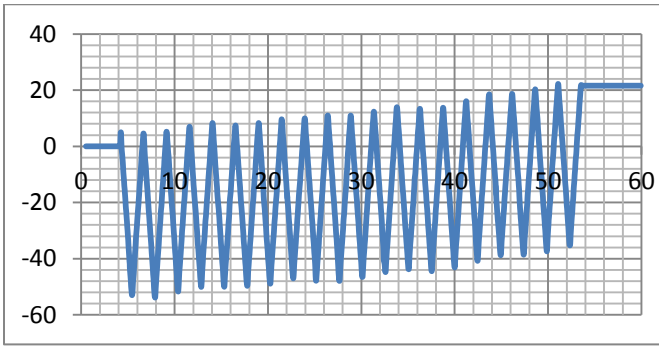


Figure 5.4: Random sample from Test 2.

Test 3. 50 cycles.
Time average = 125.88 [s]. Angle displacement average = 61.6 [deg]. Drift average = 0.4893 [deg/s]

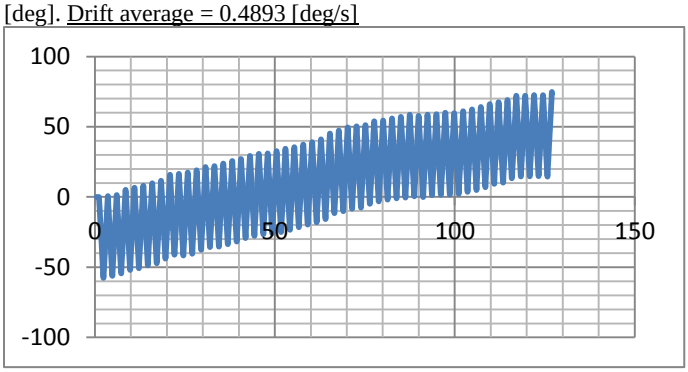


Figure 5.5: Random sample from Test 3.

measure	time [s]	total drift [deg]	drift speed [deg/s]
1	125.8	68	0.540540541
2	125.9	52	0.413026211
3	126	76	0.603174603
4	125.8	58	0.461049285
5	125.9	54	0.428911835
avg	125.88	61.6	0.489340495

Table 5.2: Other results from Test 3.

The complete spreadsheets that contain all the data collected throughout this set of experiments can also be found in the files attached to this project.

2. SIMULATED RESULTS

The algorithm was first tested creating a simulation of the expected performances of the sensors. The simulation was expected to perform the same operation as the drift experiment; this is rotating about the z-axis counter clockwise and clockwise at a constant speed in a total angle difference of 60 degrees. This data could be considered as the input for the algorithm as is the "real" movement of the platform. The following considerations were made to simulate the output each sensor:

Gyroscope: The output of the gyroscope was programmed to follow the original output presenting an increase as times goes by. Even though the output of the gyro presents, according to the data obtain in the gyro drift experiment, a drift rate of about 0.5 deg/s, in order to be able to appreciate the effect of the drift in really short times, some of the simulation were performed utilizing a drift rate of 10 deg/s.

Camera: The camera output was designed to follow with complete accuracy the input angle, but with a delay between each sample of 33 milliseconds, simulating the sampling rate of the silicon retina. This effect looks in the graphs as an under-sampled output of the angle input.

An example of the simulation performed can be observed in Figure 5.6.

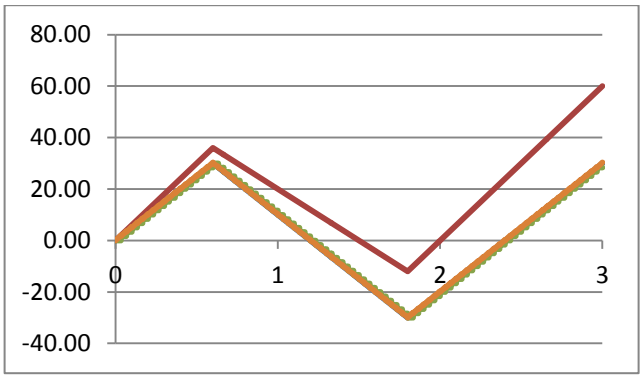


Fig. 5.6: Simulation of gyro output with drift (red) camera output with slow sampling rate (green) and the combined output (yellow)

In Figure 5.7 below we can observe in greater detail the first 100 milliseconds of the simulation presented in Figure 5.6. Here it is pretty evident how the output of the algorithm in yellow, tries to follow the output of the gyro but gets compensated each time the value of the camera changes, therefore removing successfully the drift of the gyro and combining the best of both technologies into one.

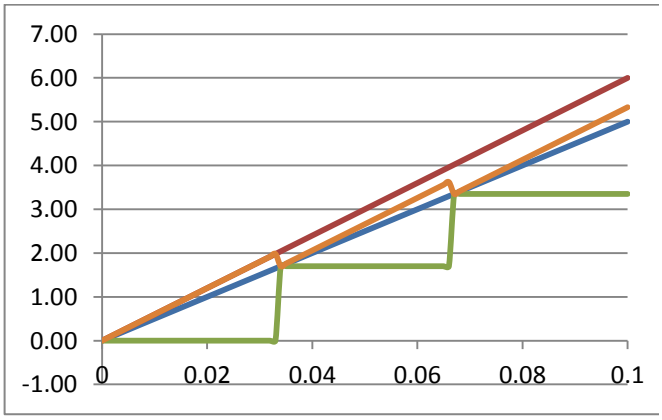


Fig. 5.7: Zoom in to the first few milliseconds, we can observe all the outputs from Fig. 1 plus the real angle of the system (blue)

Finally, in Figure 5.8 we can observe a simulation ran over seven seconds using the drift value obtained from the drift experiments. Since we already observed that the simulation performed quite well even with a drift of 10 degrees per seconds, this figure serves only to prove that the algorithm would perform well under realistic conditions.

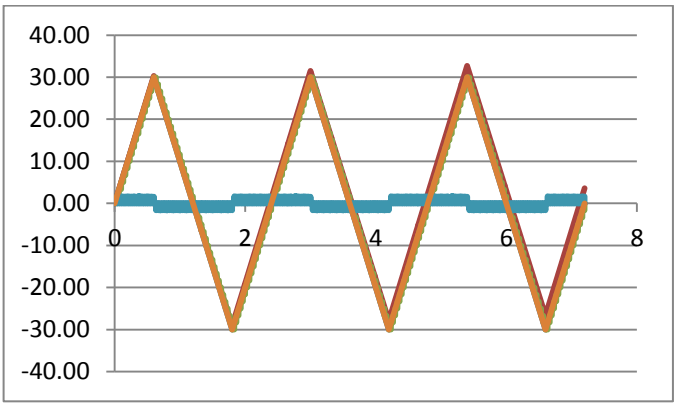


Fig. 5.8: gyro output with an approximate of the real drift according to the performed experiments.

3. REAL RESULTS

Uploading the algorithm explained in Section IV of this paper, the algorithm was tested in the experimental platform. Figure 5.9 below shows a picture of the platform in the test site.

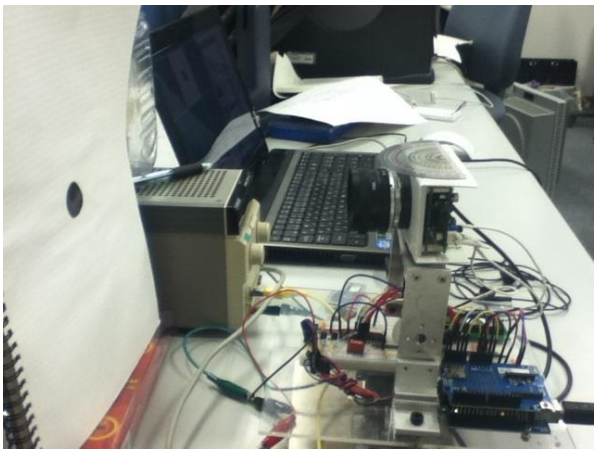


Fig.5.9: Picture of the whole experimental system: A computer (up), a power source (middle), a black dot in a white background as a target (left) and the rotating platform with the sensors (right).

The output of the gyro and the camera can be appreciated on Figure 5.10. We can observe that the angle detected by both sensors is the same but in the opposite direction, since the angle of the gyro is the angle of the real body and the angle from the camera is the visual angle taken from the target in the visual field.

We can also observe a lot of noise in the camera output (on red) this might be due to a glitch in communication between the PSoC and the FPGA and can be solved debugging the code to improve the performance of the system. While it is a noise that cannot be ignored for a high sensitivity inertial application like this, since most of the data sampled is along a certain function the servo motor doesn't have time to react to this order so it is simply ignored.

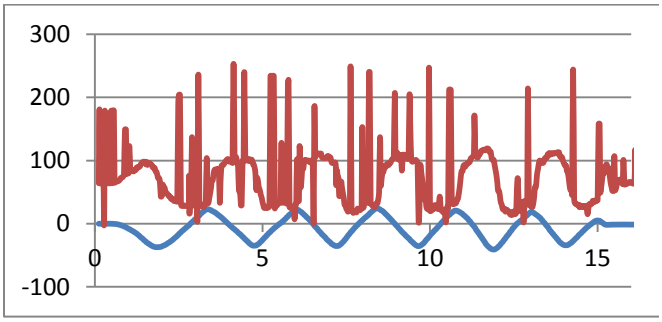


Fig. 5.10. Output data of gyro (blue) and camera (red)

To see a video of the experimental results moving the motor using only the input from the gyro, only the input from the camera and the combined result refer to the files in the attached folder.

VI. Conclusions and Future works

Gaze stabilization is a topic that has been on research for the past 30 years and many researchers have used different approaches to successfully counteract the movement caused by inertia or other effects. The current paper presented an approach using visual feedback as a main supply for compensation with the aid of the gyroscope to make up for the time delays between each sample of the camera.

The performance of the algorithm was proven successful both in the simulation and in the real experimentation. The only problem I find is that the system is too restricted to be able to have a practical application as is.

While this project performed successfully, the restrictions of the system were too heavy to consider it practical. A list of improvements that can be performed to the system goes as follows:

- ✓ Increase degrees of freedom: Currently, the project was designed to compensate the movement in one direction, in the x-axis, or from left to right. The scope of it can be expanded to cover also vertical movement (across the y-axis) and even rotational movement.
- ✓ Integrate other inertial sensors: The integration of sensors such as an accelerometer, a magnetometer or a GPS Antenna would give us further information about the environment, the position of the object with more degrees of freedom and in where direction it is moving with exceeding precision, which would help to raise the accuracy of the output.
- ✓ Perform further tests: To be able to secure the practical performance of this system in real life applications several tests must be performed to ensure that, for example, linear movement in an accelerating car will not affect the output of the other sensors significantly.
- ✓ Increase robustness of object tracking: One of the most important and practical improvements that can be done to the project is make a visual system that can track objects in the real world. Since the applications include navigation the system could be configured to be able to recognize and follow the lines of a road, therefore making for a more stable image of the environment surrounding the car to have further application in obstacle detection and safety measures.

Acknowledgments

Thanks to all the members of Yagi Laboratory (BioSystems and Devices Area, Division of Electrical, Electronic and Information Engineering, Graduate School of Engineering, Osaka University) for helping me throughout this project whenever I needed. Thanks to Osaka University for allowing me to use its installations at free will. Thanks to Universidad Nacional Autonoma de Mexico for promoting international exchange and administering scholarships.

References

[1] P. Corke, J. Lobo and J. Dias, "An Introduction to Inertial and Visual Sensing" CSIRO ICT Centre, Brisbane, Australia and University of Coimbra, Portugal. June 2007.

[2] T. Shibata and S. Schaal, " Biometric Gaze Stabilization" Kawato Dynamic Brain Project, ERATO, JST and USC

[3] T. Yagi, S. Kameda. "An Analog VLSI Chip Emulating Sustained and Transient Response channels of the Vertebrate Retina" IEEE transactions on neural networks, vol-14, no. 5, September 2003.

[4] G. Welch and G. Bishop, "An Introduction to the Kalman Filter" University of North Carolina at Chapel Hill, Chapel Hill, NC 27599-3175. July, 2006

[5] P. Albertos, A. Crespo, "Real Time Control of Non Uniformly Sampled Systems" Dept. of Systems Eng., Computers and Control Universidad Politécnica de Valencia, Spain

[6] F. Haugen "Discrete Time Signals and Systems" TechTeach, February, 2005

[7] V. Toochinda "Digital PID Controllers" June 2011, can be found at: http://www.controlsystemslab.com

[8] "CY8C29466 – PSoC™ Mixed Signal Array" *August 2004, Cypress MicroSystems, Inc.*

[9] "±1.5g, ±6g Three Axis Low-g Micromachined Accelerometer" *April 2008, Freescale Semiconductor Inc.*

[10] "ITG-3200 Product Specification, Revision 1.7" *February 2011, InvenSense Inc.*

[11] "LM150/LM350A/LM350 - 3-Amp Adjustable Regulators." *May 1998, National Semiconductor Corporation*

[12] D. Van Ess, "AN2096 - PSoC™ 1, Using the ADCINC Analog to Digital Converter" June, 2011

[13] D. Van Ess, "AN2041 - PSoC™ 1, Understanding Switched Capacitor Filters" September, 2011

[14] T. Dust, "AN50987 - Getting Started with I^2C in PSoC™ 1" November, 2011

[15] R. Klavon, "ECE480 - UART Communication with the PSoC Development Kit" March, 2011

[16] T. Dust "AN51234 - Getting Started with SPI on PSoC 1™" Nobember, 2011

[17] UM10204 – I2C Bus Specification and User Manual. Can be found at: http://www.nxp.com/documents/user_manual/UM10204.pdf

[18] David Kalinsky and Roee Kalinsky, "Introduction to Serial Peripheral Interface" January 2002, can be found at: http://www.eetimes.com/discussion/beginner-s-corner/4023908/Introduction-to-Serial-Peripheral-Interface

[19] Starlino, "A guide to using IMU (Accelerometer and Gyroscope Devices) in Embedded Applications", December, 2009. Can be found at http://www.starlino.com/imu_guide.html

[20] M. Kraft "CLOSED LOOP DIGITAL ACCELEROMETER EMPLOYING OVERSAMPLING CONVERSION, Chapter 2. Principle of operation of an accelerometer" Coventry University, School of Engineering, UK, 1997.

APÉNDICE B

CÓDIGO COMPLETO EN LENGUAJE C++ UTILIZADO PARA LA PLATAFORMA ARDUINO UNO

```
// ------------------------------------------------------------------------
---
// LOGGER (Ver. 0.7)
// Designer: Edwinn Gamborino
//
// This program performs the following operations:
//
// - Read tri-axial data from the ITG3200 digital gyroscope via the I2C protocol
// - Perform operations to obtain the angular position about the z axis
// - Sand and receive data via I2C protocol with a PSoC
// - Display data in the computer via the Serial Monitor
// - Save the same data in an SD card as a CSV file for later analysis
// ------------------------------------------------------------------------
---

//-----------------------------------------------------
// Libraries and variables and constants
//-----------------------------------------------------

#include <SD.h>
#include <Wire.h>

// Accelerometer inputs
#define ACC_Y 0            // analog 0
#define ACC_X 1            // analog 1
#define ACC_Z 2            // analog 2

// Registers in the ITG-3200
char WHO_AM_I = 0x00;
char SMPLRT_DIV= 0x15;
char DLPF_FS = 0x16;
char GYRO_XOUT_H = 0x1D;
char GYRO_XOUT_L = 0x1E;
char GYRO_YOUT_H = 0x1F;
char GYRO_YOUT_L = 0x20;
char GYRO_ZOUT_H = 0x21;
char GYRO_ZOUT_L = 0x22;

// Settings for the gyro
char DLPF_CFG_0 = 1<<0;
char DLPF_CFG_1 = 1<<1;
char DLPF_CFG_2 = 1<<2;
```

```
char DLPF_FS_SEL_0 = 1<<3;
char DLPF_FS_SEL_1 = 1<<4;

// Constant values
const int chipSelect = 4;
File logfile;
#define SAMPLING_RATE 1      // mills between entries
#define SYNC_INTERVAL 1000   // mills between calls to write data to the card

uint32_t syncTime = 0;       // time of last sync()
char itgAddress = 0x69;      // ITG3200 address
char psocAddress = 0x04;     // PSoC address
byte b,c,d;

//------------------------------------------------------------
// Setup
//------------------------------------------------------------

void setup(void)
{
  // Wake up serial bus
  Serial.begin(9600);
  Serial.println();

  //Initialize I2C communication.
  Wire.begin();

  //Read the WHO_AM_I register
  char itgID = 0, psocID = 0;
  itgID = itgRead(itgAddress, 0x00);

  //Read the PSoC address
  Wire.beginTransmission(psocAddress);
  Wire.requestFrom(psocAddress, 3);
  if(Wire.available()){
    b = Wire.read();
    c = Wire.read();
    d = Wire.read();
}
Wire.endTransmission();

  // Print both addresses on the serial monitor
  Serial.println("Initializing I2C bus... finding slaves...");
  Serial.print("ITG ID: ");
```

```
Serial.println(itgID, HEX);
Serial.print("PSoC ID: ");
Serial.println(d, HEX);

//Configure the gyroscope
//Set the gyroscope scale for the outputs to +/-2000 degrees per second
itgWrite(itgAddress, DLPF_FS, (DLPF_FS_SEL_0|DLPF_FS_SEL_1|DLPF_CFG_0));
//Set the sample rate to 100 hz
itgWrite(itgAddress, SMPLRT_DIV, 9);

// initialize the SD card
Serial.print("Initializing SD card...");
// make sure that the default chip select pin is set to output
pinMode(10, OUTPUT);

// see if the card is present and can be initialized:
if (!SD.begin(chipSelect)) {
  error("Card failed, or not present");
}
Serial.println("card initialized.");

// create a new file
char filename[] = "LOGGER00.CSV";
for (uint8_t i = 0; i < 100; i++) {
  filename[6] = i/10 + '0';
  filename[7] = i%10 + '0';
  if (! SD.exists(filename)) {
    // only open a new file if it doesn't exist
    logfile = SD.open(filename, FILE_WRITE);
    break;  // leave the loop!
  }
}

// if the logfile couldnt be found, print error
if (! logfile) {
  error("couldnt create file");
}

// print on the serial terminal the filename
Serial.print("Logging to: ");
Serial.println(filename);

// print the headers of the columns
logfile.print ("time [ms]");
```

```
  logfile.print(", ");
  logfile.print("z [deg/s]");
  logfile.print(", ");
  logfile.print("theta_g [deg]");
  logfile.print(", ");
  logfile.println("theta_c [deg]");
  Serial.println ("time [ms], theta_g[deg], theta_c [deg] ");
}

//-------------------------------------------------------------
// Main loop
//-------------------------------------------------------------

 void loop (void){

 //Create variables to hold the output rates.
 int xRate, yRate, zRate;
 static unsigned long newMicros, lastMicros, interval;
 static float zAvg, zLast, theta, thetaLast;

 // delay for the amount of time we want between readings
 delay((SAMPLING_RATE -1) - (millis() % SAMPLING_RATE));

 // log milliseconds since starting
 float m = millis();
 m = m/1000;
 logfile.print(m);         // milliseconds since start
 Serial.print(m);         // milliseconds since start

 // get time interval
 lastMicros = newMicros;
 newMicros = micros();
 interval = newMicros - lastMicros;

 // Read the data to log and present
 float acc_x = analogRead(ACC_X);
 float acc_y = analogRead(ACC_Y);
 float acc_z = analogRead(ACC_Z);

 // Convert to g
 acc_x = (((acc_x*5/1023)-1.65)/.8)-.25;
 acc_y = (((acc_y*5/1023)-1.65)/.8);
 acc_z = (((acc_z*5/1023)-1.65)/.8)+.28;
```

```
 //Read the x,y and z output rates from the gyroscope.
 xRate = readX();
 yRate = readY();
 zRate = readZ();

 // Convert to deg/s
 xRate = xRate/14.375;
 yRate = yRate/14.375;
  zRate = zRate/14.375;

  //Angle ゲット
  zAvg = (zRate+zLast)/2;
  theta = thetaLast + (zAvg*interval)/1000000;

  //Last values ゲット
  zLast = zRate;
  thetaLast = theta;

  // Get value from camera
  int cam = readPSoC();
  int cam_r = readPSoC2();

  //Print data in both SD and Serial Monitor
  logfile.print(", ");
  logfile.print(zAvg);
  logfile.print(", ");
  logfile.print(theta);
  logfile.print(", ");
  logfile.print(cam);
  Serial.print(", ");
  Serial.print(zAvg);
  Serial.print(", ");
  Serial.print(theta);
  Serial.print(", ");
  Serial.print(cam);
  logfile.println();
  Serial.println();

  // Convert the angular position to integers and split it into 8-bit constants
to send it via I2C.
  float thetaint = theta*100;
  int trans = (int) thetaint;
  uint8_t transLow = trans & 0xff;
  uint8_t transHigh = (trans >> 8);
```

```
  //Send the angular position to PSoC
  Wire.beginTransmission(psocAddress);
  Wire.write(transLow);
  Wire.write(transHigh);
  Wire.endTransmission();

  // Now we write data to disk! Don't sync too often - requires 2048 bytes of I/O
to
  // SD card which uses a bunch of power and takes time
  if ((millis() - syncTime) < SYNC_INTERVAL) return;
  syncTime = millis();
  logfile.flush();
}

//-------------------------------------------------------
// Complementary functions
//-------------------------------------------------------

// Error routine
void error(char *str)
{
  Serial.print("error: ");
  Serial.println(str);

  while(1);
}

//Write function
void itgWrite(char address, char registerAddress, char data)
{

  Wire.beginTransmission(address);
  Wire.write(registerAddress);
  Wire.write(data);
  Wire.endTransmission();
}

//Read function
unsigned char itgRead(char address, char registerAddress)
{
  unsigned char data=0;

  Wire.beginTransmission(address);
```

```
  Wire.write(registerAddress);
  Wire.endTransmission();

  //Ask the I2C device for data
  Wire.beginTransmission(address);
  Wire.requestFrom(address, 1);

  //Wait for a response from the I2C device
  if(Wire.available()){
    data = Wire.read();
  }
  Wire.endTransmission();
  return data;
}

//Functions to read and stack the 16-bit values from gyro
int readX(void)
{
  int data=0;
  data = itgRead(itgAddress, GYRO_XOUT_H)<<8;
  data |= itgRead(itgAddress, GYRO_XOUT_L);

  return data;
}
int readY(void)
{
  int data=0;
  data = itgRead(itgAddress, GYRO_YOUT_H)<<8;
  data |= itgRead(itgAddress, GYRO_YOUT_L);

  return data;
}
int readZ(void)
{
  int data=0;
  data = itgRead(itgAddress, GYRO_ZOUT_H)<<8;
  data |= itgRead(itgAddress, GYRO_ZOUT_L);

  return data;
}

//Read the PSoC address
  int readPSoC (void)
  {
```

```
    int a,b,c,data=0;
    Wire.beginTransmission(psocAddress);
    Wire.requestFrom(psocAddress, 4);
    if(Wire.available()){
      a = Wire.read();
      b = Wire.read();
      c = Wire.read();
      data = Wire.read();
    }
    Wire.endTransmission();
    return data;
  }

int readPSoC2 (void)
  {
    int a,b,c,d,data=0;
    Wire.beginTransmission(psocAddress);
    Wire.requestFrom(psocAddress, 5);
    if(Wire.available()){
      a = Wire.read();
      b = Wire.read();
      c = Wire.read();
      d = Wire.read();
      data = Wire.read();
    }
    Wire.endTransmission();
    return data;
  }
```

APÉNDICE C

CÓDIGO COMPLETO EN LENGUAJE C++ UTILIZADO PARA LA PLATAFORMA PSOC 1

```
//-------------------------------------------------------------------------
// Servo Final Arduino Test.
//
// This program is designed as a test for the final application of the PSoC in
// the project. It will perform the following operations simultaneously:
// - Sample the 3 axes of the accelerometer using 3 ADCs
// - Sample the 3 axes of the gyroscope using I2C communication protocol
// - Communicate with an Arduino board via SPI protocol acting as a slave
// - Perform filtering operations on the input from the sensors
// - Send the resulting data to the master
// - Print the resulting data in a LCD module
//-------------------------------------------------------------------------

#include <m8c.h>
          // part specific constants and macros
#include <PSoCAPI.h>
#include <stdlib.h>
        // Libraries to write int and float to LCD
#include <string.h>
#include <math.h>

#define PI 3.14159265

/* Write/Read buffers */
char X_OUT[2];
char Y_OUT[2];
char Z_OUT[2];
char T_OUT[2];
char X_ADD[2];
char Y_ADD[2];
char Z_ADD[2];
char T_ADD[2];
char SMPLRT_DIV[2];
char DLPF_CS[2];

double cRxData;
char cStatus;
int dong;

// 8 bit variables, for the accelerometer an gyro (16 bit in total each)
char ACC_Xhi, ACC_Xlo, ACC_Yhi, ACC_Ylo, ACC_Zhi, ACC_Zlo;
int GYRO_XZ, GYRO_YZ, GYRO_XY, ACC_X, ACC_Y, ACC_Z, ACC_R;
// Variables to store the raw acc angles.
int ACC_AX, ACC_AY, ACC_AZ;
```

```
// Vector to store the position estimate.
float R_EX, R_EY, R_EZ;
// Vector to store the angle from gyro.
float R_GX, R_GY, R_GZ, PWM_X, PWM_Y;
int k1,k2;

/* ITG3200 Registers */
// Contains the I2C address of the sensor, 0x69 by default.
char WHO_AM_I   = 0x00;
// Sample rate divider, determines the sample rate of the gyros.
char SMPLRT_REG = 0x15;
// Configuration of filter and power register.
char DLPF_REG   = 0x16;
void Delay(void);

void main()
{
/* The scale setting (FS_SEL), must be 0x03 for proper operation. (1 in bits 4
and 3) and the Digital low pass filter configuration (DLPF_CFG) must be set to 3
(011 in bits 2 to 0) for 1 kHz sample rate and 42 Hz bandwidth. Therefore, the
value that must be sent to the DLPF_FS register is b 0001 1011 = 0x1B;
The sample rate is set to 9 for 100 Hz sampling frequency. */

      SMPLRT_DIV[0] = SMPLRT_REG;
      SMPLRT_DIV[1] = 0x09;
      DLPF_CS[0]    = DLPF_REG;
      DLPF_CS[1]    = 0x1B;

      /* Register addresses for X,Y,Z axes and temperature */
      X_ADD[0]      = 0x1C;
      Y_ADD[0]      = 0x1E;
      Z_ADD[0]      = 0x20;
      T_ADD[0]      = 0x1A;

      /*Enable global interrupts and start LCD module */
      M8C_EnableGInt;
      LCD_1_Start();

      /* Start I2C user module and enable its interrupts */
      I2CHW_1_Start();
      I2CHW_1_EnableInt();
      I2CHW_1_EnableMstr();

      /* Start SPIS user module and enable its interrupts */
```

```
SPIS_1_Start(SPIS_1_SPIS_MODE_0);
SPIS_1_EnableInt();
/* Start followers */
PGA_1_Start(PGA_1_HIGHPOWER);
PGA_2_Start(PGA_2_HIGHPOWER);
PGA_3_Start(PGA_3_HIGHPOWER);

/* Start PWMs */
PWM16_1_Start();
PWM16_2_Start();
PWM8_1_Start ();

/* Apply power to the ADC blocks and get samples from all three */
ADCINC12_1_Start(ADCINC12_1_HIGHPOWER);
ADCINC12_2_Start(ADCINC12_2_HIGHPOWER);
ADCINC12_3_Start(ADCINC12_3_HIGHPOWER);
ADCINC12_1_GetSamples(0);
ADCINC12_2_GetSamples(0);
ADCINC12_3_GetSamples(0);

/* Initialize SPI_TX register with 0 */
SPIS_1_SetupTxData(0x00);

/* Write sample rate divisor to the gyro. */
I2CHW_1_bWriteBytes(0x69, SMPLRT_DIV, 2, 0);
// Wait for the operation to complete
while (!(I2CHW_1_bReadI2CStatus() & I2CHW_WR_COMPLETE));
// Clear the Write Status
I2CHW_1_ClrWrStatus();

/* Write low pass filter configuration */
I2CHW_1_bWriteBytes(0x69, DLPF_CS, 2, 0);
while (!(I2CHW_1_bReadI2CStatus() & I2CHW_WR_COMPLETE));
I2CHW_1_ClrWrStatus();

/* Main routine */
for(;;)
{
// if P0[0] (slave select) is set, enter routine
if (PRT0DR & 0x01)
            {
      /* Read ADCs */
      // Loop until value ready
      while(ADCINC12_1_fIsDataAvailable() == 0);
```

```
// Clear ADC flags
ADCINC12_1_ClearFlag();
ADCINC12_2_ClearFlag();
ADCINC12_3_ClearFlag();

// Get ADC results
ACC_Y = ADCINC12_1_iGetData();
ACC_X = ADCINC12_2_iGetData();
ACC_Z = ADCINC12_3_iGetData();

/* Adjust ADC offset */
ACC_X = ACC_X+2048;
ACC_Y = ACC_Y+2048;
ACC_Z = ACC_Z+2048;

/* Read I2C bus and save values */
// Read X axis.
itgRead(X_ADD, X_OUT);
// Read Y axis.
itgRead(Y_ADD, Y_OUT);
// Read Z axis.
itgRead(Z_ADD, Z_OUT);
// OR XH and XL in a single value
GYRO_XZ = X_OUT[0]<<8|X_OUT[1];
// OR YH and YL in a single value
GYRO_YZ = Y_OUT[0]<<8|Y_OUT[1];
// OR ZH and ZL in a single value
GYRO_XY = Z_OUT[0]<<8|Z_OUT[1];

/* Divide the measurements in 8 bit values to send via SPI */
ACC_Xhi = (ACC_X>>8)&255;
ACC_Xlo = (ACC_X&255);
ACC_Yhi = (ACC_Y>>8)&255;
ACC_Ylo = (ACC_Y&255);
ACC_Zhi = (ACC_Z>>8)&255;
ACC_Zlo = (ACC_Z&255);

      /* Send the data via SPI to Arduino */
      cStatus = SPIS_1_bReadStatus();
      if(cStatus & SPIS_1_SPIS_RX_BUFFER_FULL)
            {
                  // Read the value that the master wants to read
                  dong = SPIS_1_bReadRxData();
```

```
                        // If its 0x00, send ACC_X
                        if (dong == 0xAA)
                        {
                              SPIS_1_SetupTxData(ACC_Xhi);
                              SPIS_1_SetupTxData(ACC_Xlo);
                        }
                        // If its 0x01, send ACC_Y
                        else if (dong == 0xBB)
                                    {
                              SPIS_1_SetupTxData(ACC_Yhi);
                              SPIS_1_SetupTxData(ACC_Ylo);
                        }
                        // If its 0x02, send ACC_Z
                        else if (dong == 0xCC)
                        {
                              SPIS_1_SetupTxData(ACC_Zhi);
                              SPIS_1_SetupTxData(ACC_Zlo);
                        }
                        // If its 0x03, send GYRO_XZ
                        else if (dong == 0xDD)
                              {
                              SPIS_1_SetupTxData(X_OUT[0]);
                              SPIS_1_SetupTxData(X_OUT[1]);
                        }
                        // If its 0x04, send GYRO_YZ
                        else if (dong == 0xEE)
                        {
                              SPIS_1_SetupTxData(Y_OUT[0]);
                              SPIS_1_SetupTxData(Y_OUT[1]);
                        }
                        // If its 0x05, send GYRO_XY
                        else if (dong == 0xFF)
                              {
                              SPIS_1_SetupTxData(Z_OUT[0]);
                              SPIS_1_SetupTxData(Z_OUT[1]);
                        }
                        // If none of the prior, just send zero
                        else
                              {
                              SPIS_1_SetupTxData(0x00);
                        }
                  }

      /* Print values on LCD */
```

```
            LCD_1_Position(0,0);
            LCD_1_PrHexInt(ACC_X);
            LCD_1_Position(0,6);
            LCD_1_PrHexInt(ACC_Y);
            LCD_1_Position(0,12);
            LCD_1_PrHexInt(ACC_Z);
            LCD_1_Position(1,0);
            LCD_1_PrHexInt(GYRO_XZ);
            LCD_1_Position(1,6);
            LCD_1_PrHexInt(GYRO_YZ);
            LCD_1_Position(1,12);
            LCD_1_PrHexInt(GYRO_XY);
            }
      }
}

/* Routine to write float point values in LCD module. */
void LCD_1_PrintInt(float val)
{
      int str;
      char* buf;
      buf = ftoa(val,&str);
      LCD_1_PrString(buf);
}

/* Routine to read any register in the gyro. */
void itgRead (char reg[2], char data[2])
{
      // Send the register form which we want to read.
      I2CHW_1_bWriteBytes(0x69, reg, 2, 0);
      // Wait for the operation to complete.
      while (!(I2CHW_1_bReadI2CStatus() & I2CHW_WR_COMPLETE));
      // Clear the Write Status.
      I2CHW_1_ClrWrStatus();
      // Read the information from said register.
      I2CHW_1_fReadBytes(0x69, data, 2, 0);
      // Wait for the operation to complete.
      while (!(I2CHW_1_bReadI2CStatus() & I2CHW_RD_COMPLETE));
      // Clear the Read Status.
      I2CHW_1_ClrRdStatus();
}
```