Trabajo Fin de Carrera

Enginyeria Tècnica de Telecomunicació
Especialitat en Sistemes Electrònics

Desarrollo de un Sistema Integrado de Navegación Inercial: Interfície IMU + FPGA

Alfred Raul Giménez Bonastre

Director: Carles Ferrer Ramis

Departamento de Microelectrónica y Sistemas Electrónicos

Asesor: Alex Garcia Quinchia

Departamento de Microelectrónica y Sistemas Electrónicos

Escola Tècnica Superior d'Enginyeria (ETSE)

Universitat Autònoma de Barcelona (UAB)

Julio 2010

UAB

El tribunal de avaluación de este Trabajo Final de Carrera, reunido el día 7 de Julio de 2010, ha acordado conceder la siguiente cualificación:

Presidente:

Vocal:

Secretari:

UAB

El abajo firmante, *Carles Ferrer Ramis*, Profesor de l'Escola Tècnica Superior d'Enginyeria (ETSE) de la Universitat Autònoma de Barcelona (UAB),

CERTIFICA:

Que el trabajo presentado en esta memoria de Trabajo Final de Carrera ha sido realizado bajo su dirección por el alumno *Alfred Raul Giménez Bonastre*.

Y, para que conste a todos los efectos, firma el presente certificado.

Bellaterra, *20 de Mayo de 2010*.

Firma: *Carles Ferrer Ramis*

ÍNDICE

1. Introducción 9

1.1 Descripción de la Situación a Tratar 10

1.2 Objetivos 11

2. Fundamentos Teóricos 13

2.1 Unidad de Medida Inercial 13

2.1.1. Componentes de una IMU 13

2.1.2. Caracterización de una IMU 15

2.2 Filtro de Kalman 16

2.3 FPGA 19

2.3.1. Historia de la FPGA 20

2.3.2. Aplicaciones 21

2.3.3. Arquitectura 22

2.3.4. Procesador PowerPC 405 23

3. Descripción General del Proyecto 24

3.1 IMUs del Mercado 24

3.1.1. IMU 6DOF Razor - Ultra-Thin IMU 24

3.1.2. Atomic IMU 6 Degrees of Freedom - XBee Ready 25

3.2 IMU Adquirida 26

3.2.1. Comparación *Atomic IMU* con la unidad de la ETSE 26

3.3 Funcionamiento y Programación de la Atomic IMU 27

3.3.1. Diagramas de Tiempo 31

3.4 Datos obtenidos de la IMU y cálculo del ángulo. Filtro de Kalman 33

3.4.1. Aceleración 33

3.4.2. Ángulo a partir de la aceleración 34

3.4.3. Velocidad Angular 36
3.4.4. Ángulo a partir de la velocidad angular 36
3.4.5. Filtro de Kalman. Resultados 37
3.5 Programación FPGA 41
3.6 Programa en Visual Basic 43
4. Conclusiones y líneas futuras 45
5. Referencias Bibliográficas 46
Anexo A 47
Anexo B 54
Anexo C 65
Anexo D 68

ÍNDICE DE FIGURAS

Figura 1. Arquitectura general de la integración GPS/INS. 10

Figura 2. Arquitectura general de la integración GPS/INS con corrección de ángulo (IMU) mediante el filtro de Kalman. 11

Figura 3. Vehículo Aéreo no Tripulado (UAV). 12

Figura 4. Orientación proporcionada por una IMU. 13

Figura 5. Ciclo del filtro de Kalman. 17

Figura 6. Visión completa del filtro de Kalman. 19

Figura 7. Arquitectura interna de una FPGA. 20

Figura 8. FPGA de Xilinx - Virtex-II Pro. 21

Figura 9. Ejemplo simple de una celda lógica. 22

Figura 11. IMU 6DOF Razor. 24

Figura 12. *Atomic IMU 6DOF – XBee Ready*. 25

Figura 13. Conversor MAX232. 28

Figura 14. Conversor MAX232 y el circuito acondicionador montado. 29

Figura 15. Cable para conectar el MAX232 con el PC mediante RS-232. 29

Figura 16. Programador Pony-Prog utilizado. 30

Figura 17. Vista superior de la *Atomic IMU*. 31

Figura 18. Diagrama de tiempo para una frecuencia de reloj de 1 MHz y una frecuencia de trabajo de 100 Hz. 32

Figura 19. Diagrama de tiempo para una frecuencia de reloj de 10 MHz y una frecuencia de trabajo de 100 Hz. 32

Figura 20. Datos obtenidos del acelerómetro y aceleración (en el eje x) para f = 150 Hz. 34

Figura 21. Ángulo pitch a partir del acelerómetro y la gravedad. 34

Figura 22. Ángulo pitch obtenido a partir de la aceleración para f = 150 Hz. 35

Figura 23. Datos obtenidos del sensor y velocidad angular (pitch) para f = 150 Hz. 36

Figura 24. Velocidad angular y ángulo pitch obtenidos a partir de los datos del giróscopo para f = 150 Hz. 37

Figura 25. Ángulo pitch sin filtro y con filtro de Kalman para f = 150 Hz. 38

Figura 26. Ángulo pitch sin filtro y con filtro de Kalman para f = 150 Hz. 39

Figura 27. Ángulo pitch sin filtro y con filtro de Kalman para f = 200 Hz. 39

Figura 28. Ángulo pitch sin filtro y con filtro de Kalman para f = 250 Hz. 40

Figura 29. Ángulo roll sin filtro y con filtro de Kalman para f = 200 Hz. 40

Figura 30. Diagrama de bloques de la conexión PLB de la FPGA con los controladores GPIO y UARTLITE. 42

Figura 31. *Atomic Mixer* proporcionado por *Sparkfun*. 43

Figura 32. Programa *Atomic Mixer* Adaptado. 44

Figura 10. Arquitectura del sistema con PowerPC. 23

1.Introducción

Cada vez encontramos más aplicaciones automatizadas que requieren de un control de movimiento o estabilidad para poder llevar a cabo su tarea. Por ejemplo, si queremos enviar un satélite al espacio debemos controlar, de forma muy precisa, su movimiento hasta el espacio. Para ello se utilizan sistemas de medida inercial formados por diferentes componentes como una unidad de medida inercial, también conocida como IMU[1].

La unidad de medida inercial es el componente principal de estos sistemas de guía inerciales. Trabaja gracias a unos sensores específicos tales como acelerómetros y giróscopos debidamente colocados en una placa, que proporcionan al usuario datos como la aceleración y rotación, en los tres ejes de coordenadas, que sufra dicha placa. Normalmente viene acompañada de un receptor GPS[2] para tener más exactitud en las medidas y del cual extraen la posición inicial para así poder calcular el recorrido efectuado.

Éstas se utilizan en mayor medida para guiar un objeto, ya sea un avión no tripulado o UAV[3] o hasta incluso un robot. Todos éstos tienen en común que necesitan de un sistema de guía inercial para poder llevar a cabo su función. También son muy utilizadas en lanzaderas y satélites, ya que conociendo su movimiento pueden corregir su ruta y así guiarse automáticamente sin necesidad de intervención humana.

Actualmente se están utilizando en sistemas de posicionamiento global para tener más exactitud que la proporcionada por el GPS, aunque el precio de los receptores aumenta considerablemente.

Este trabajo tiene como finalidad la adquisición de datos de una unidad de medida inercial, también conocida como IMU a través de una FPGA[4], la cual se encargará de recibir y procesar los datos de la IMU y con la cual el usuario podrá definir la sensibilidad y la frecuencia de trabajo de la misma. Veremos como programar la unidad

---

1 Inertial Measurement Unit (Unidad de Medida Inercial).
2 *Global Positioning System* (Sistema de Posicionamiento Global).
3 Unmanned Aerial Vehicle (Vehículo Aéreo no Tripulado).
4 Field Programmable Gate Array.

de medida inercial y como se comunicará con la FPGA para obtener la mayor frecuencia de trabajo posible y aprovechar al máximo su rendimiento.

También estudiaremos el filtro de Kalman, necesario para corregir el bias producido por los giróscopos, el cuál se implementará en Matlab y en Visual Basic, donde se evaluará su rendimiento en tiempo real.

1.1 Descripción de la Situación a Tratar

Este trabajo forma parte de un proyecto de investigación del departamento de Microelectrónica y Sistemas Electrónicos de la UAB, mostrado en parte en la Figura 1, en el que se combina una unidad de media inercial con un sistema GPS para la mejora del posicionamiento proporcionado por éste, que nos ofrece un error considerable de 13 metros horizontales aproximadamente[5] y una baja frecuencia de refresco de 1 Hz. Aunque actualmente hay receptores GPS a mayor frecuencia, la mayoría de los receptores comerciales leen datos una vez por segundo. Para corregir este error se utiliza un filtro de navegación, llamado filtro de Kalman, debido a la facilidad para integrar sensores con diferente ancho de banda. A grosso modo, el uso de este filtro necesita información de la posición, velocidad y *attitude* obtenida de la IMU a través de la mecanización, para combinarla con los valores reales obtenidos del GPS. De esta forma se minimiza el error, y los datos a la salida se actualizan a la frecuencia de la unidad de medida inercial, que como veremos es mucho mayor que la del GPS (del orden de 100 – 250 Hz típicamente).

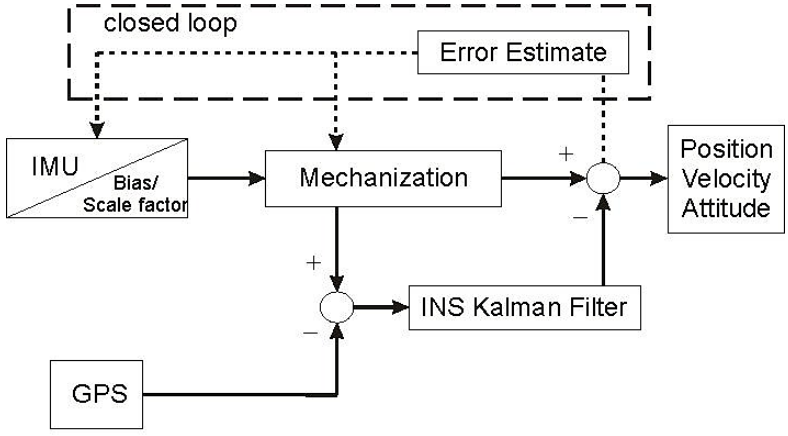

Figura 1. Arquitectura general de la integración GPS/INS.

[5] Información extraída de Integración GPS/INS: Conceptos y Experimentos.

Este trabajo se centra en la parte del proyecto referida al sistema de navegación inercial, que incluye la lectura de los datos de la IMU con la FPGA y la corrección del ángulo proporcionado por los giróscopos a través del filtro de Kalman, tal y como tenemos representado en la Figura 2.

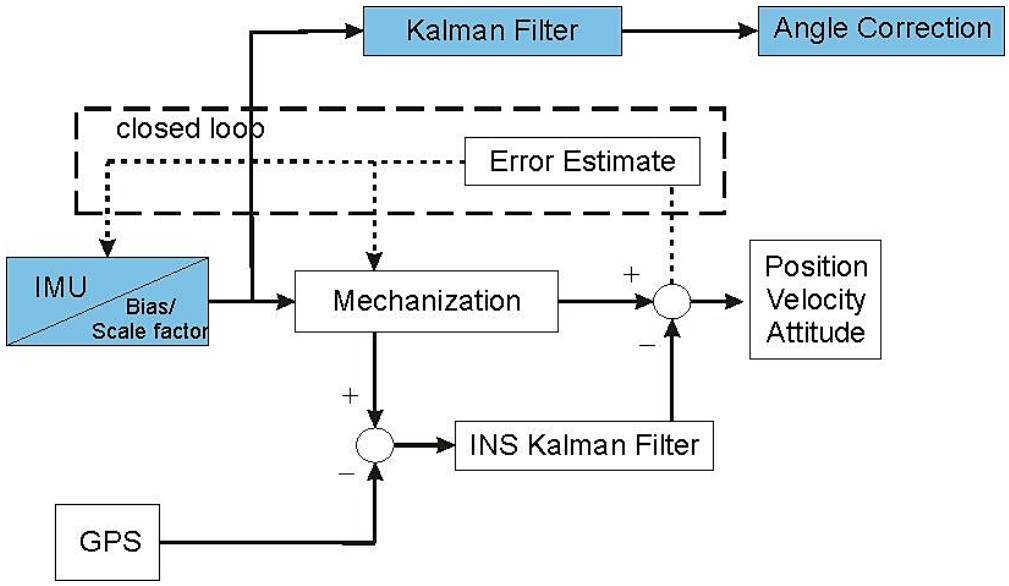

Figura 2. Arquitectura general de la integración GPS/INS con corrección de ángulo (IMU) mediante el filtro de Kalman.

1.2 Objetivos

El objetivo principal de este trabajo es realizar la lectura de aceleración y velocidad angular de la unidad de medida inercial (IMU) con la FPGA en tiempo real, además de realizar la corrección del error (bias) de los ángulos pitch y roll mediante el filtro de Kalman.

Los datos obtenidos deben ser coherentes con la realidad y lo más exactos posible, además de poder obtener una frecuencia de funcionamiento suficientemente elevada para aplicaciones UAV (Figura 3). La fácil configuración de sensibilidad y frecuencia de trabajo de la unidad también es útil si el usuario desea cambiar la aplicación o estudiar un movimiento más o menos preciso.

Para llevar a cabo este objetivo estudiaremos primero el método de trabajo de la unidad de medida inercial y la programaremos para enviar los datos a la FPGA a la mayor frecuencia posible. Se debe crear la IP de recepción y transmisión en la FPGA

para su sincronización con la IMU y programarla para poder configurar la sensibilidad y la frecuencia de la IMU mediante los *switches.*

Figura 3. Vehículo Aéreo no Tripulado (UAV).

En Matlab se estudiará el filtro de Kalman y su funcionamiento, para después implementarlo en Visual Basic junto a una aplicación que permitirá ver el comportamiento de la IMU en tiempo real.

Finalmente se comprobará que los resultados obtenidos con la corrección del error y la lectura de la IMU son correctos para su posterior aplicación en el sistema GPS/INS.

2. Fundamentos Teóricos

2.1 Unidad de Medida Inercial

Una unidad de medida inercial o IMU es un componente electrónico basado en sensores de aceleración y velocidad angular (acelerómetros y giróscopos respectivamente) la cual nos reporta el movimiento y orientación (Figura 4) que sufre dicha unidad. Es el componente principal de sistemas de guía inercial usados en vehículos aéreos, espaciales, marinos y aplicaciones robóticas.

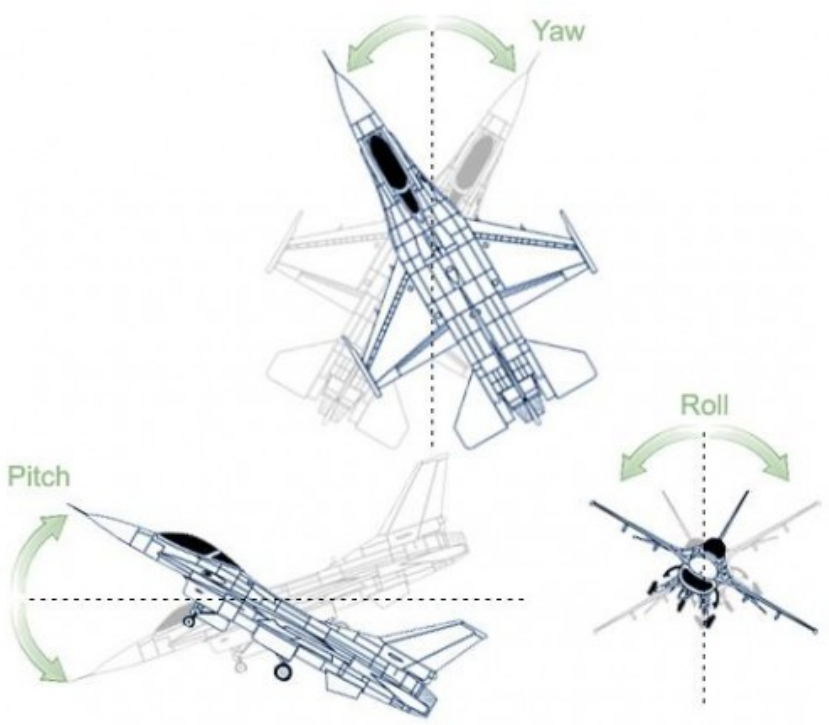

Figura 4. Orientación proporcionada por una IMU.

2.1.1. Componentes de una IMU

Cualquier unidad de medida inercial está compuesta como mínimo por un acelerómetro y un giróscopo para captar una aceleración y una velocidad angular en concreto. Generalmente, es interesante que las IMUs capten la aceleración y la velocidad angular en los tres ejes de coordenadas para conocer el movimiento exacto del componente.

También podemos encontrar unidades que incorporan un microprocesador que se encarga de recoger los datos de dichos sensores y enviarlos de forma ordenada al usuario mediante el protocolo de comunicación incorporado en la IMU.

A continuación se detallarán los componentes nombrados:

- **Acelerómetro:** instrumento capaz de medir aceleración en uno, dos o tres ejes. Existen varios tipos de acelerómetros, dependiendo de su fabricación y funcionamiento. Las IMUs incorporan acelerómetros integrados en silicio, utilizando la tecnología llamada MEMS[6], debido a la necesidad de reducir el tamaño total de la unidad. La mayoría de éstos son capacitivos, y calculan la aceleración mediante el voltaje obtenido entre dos placas una de las cuales varía su posición dependiendo del movimiento del acelerómetro. Se caracterizan por ser muy precisos en situaciones estables y tener un gran error en situaciones vibratorias o movimientos muy inestables.

- **Giróscopo:** dispositivo que mide la orientación, basándose en los principios de la conservación del momento angular. Las unidades de medida inercial utilizan giróscopos MEMS, es decir, integrados y de tamaño reducido. La salida de dicho sensor es un voltaje, la variación del cual nos indica en grados por segundo (V/º/s) la velocidad angular sufrida por el sensor. Se caracterizan por tener un error constante y lineal llamado *bias* el cual debemos tener en cuenta.

- **Microprocesador:** algunas unidades de medida inercial, como ya hemos comentado, incorporan un microprocesador. Éste es programable, pero su principal función es recoger los datos entregados por los sensores, procesarlos según desee el usuario, y enviarlos. En el microprocesador se define la frecuencia de trabajo de la unidad, que será el tiempo comprendido desde que recoge el dato del primer sensor hasta que envía al usuario el dato procesado del último sensor.

  La mayoría de los microprocesadores incorporan un conversor analógico-digital, para así convertir el voltaje dado por el sensor en una muestra. El tiempo de

[6] *Microelectromechanical Systems* (Sistemas Microelectromecánicos).

conversión influye en la frecuencia de trabajo de la unidad, como veremos más adelante.

- **Protocolo de comunicación:** los protocolos de comunicación alámbricos típicos en las IMUs son el UART, el RS-232 o el USB[7]. Algunas unidades incluyen protocolos inalámbricos, siendo los más utilizados ZigBee y Bluetooth.
- **Magnetómetro:** algunas unidades de medida inercial incluyen también sensores magnetómetros. Estos dispositivos miden la fuerza i/o dirección de los campos magnéticos que los afectan respecto el campo magnético terrestre. Aunque cabe la posibilidad de que se vean afectados por variación de otros campos magnéticos en algunas zonas.

2.1.2. Caracterización de una IMU

Los diferentes tipos de unidades de medida inercial que podemos encontrar en el mercado se caracterizan generalmente por el tipo de sensores de que están compuestas. Estos sensores, la frecuencia de trabajo, que puede interesar mayor o menor dependiendo de la aplicación para la que esté destinada la unidad, y el protocolo de comunicación son los que definen una IMU mejor que otra.

Los sensores de que se componen las unidades de medida inercial se definen principalmente por su rango de trabajo (máxima medida que soporta el sensor), su sensibilidad (relación entre la variación de la magnitud de salida y la de entrada) y su ancho de banda de respuesta (frecuencia de funcionamiento del sensor). Para el caso de los acelerómetros, el rango de trabajo se mide con la gravedad estándar, aceleración de $g = 9.80665\ m/s^2$. Podemos encontrar IMUs con un rango de 1.5g, 2g, 4g, etc. Dependiendo de este rango, obtenemos una sensibilidad u otra, ya que el nivel de voltaje máximo que podemos obtener viene definido por la alimentación del sensor. Entonces, para mayor rango, menor sensibilidad.

En los giróscopos, el rango de medida se mide en º/s, ya que obtenemos una velocidad angular, y la sensibilidad en mV/º/s. Valores típicos de rango que podemos encontrar son ± 200 º/s, ± 300 º/s, ± 500 º/s, etc.

[7] *Universal Serial Bus* (Bus Universal en Serie).

En el caso de que la IMU tenga magnetómetros, el rango se mide en gauss = 1 maxwell / cm^2.

La frecuencia de trabajo de la unidad de medida inercial viene definida por el reloj que use el microprocesador para enviar los datos al usuario y por la frecuencia de conversión del conversor analógico-digital. También influye la velocidad de transmisión, que depende del protocolo que utilicemos y de cómo lo definamos. Ésta velocidad se mide en baudios o bits por segundo y son valores típicos 38400 bps, 57600 bps, 115200 bps, etc.

2.2 Filtro de Kalman

El filtro de Kalman es la técnica de análisis por excelencia en medidas de inercia y se usa principalmente para obtener la posición y orientación de lecturas inerciales. Este filtro proporciona robustez y exactitud al sistema de medida. De esta forma, podemos utilizar sensores de más bajo precio, aunque conlleve más error.

En esta sección no detallaremos los algoritmos del filtro de Kalman, ya que es un sistema matemáticamente muy complejo, pero si comentaremos sus principios. Se puede encontrar información sobre el filtro de Kalman en [3].

El filtro de Kalman (KF) se basa en un algoritmo predictivo recursivo que utiliza el método de mínimos cuadrados. Esta solución permite calcular un estimador lineal, insesgado y óptimo del estado de un proceso en cada momento del tiempo con base en la información disponible en el instante de tiempo previo y actualizar dichas estimaciones. Se puede dividir en dos etapas tal y como nos muestra la Figura 5. La primera es la etapa de predicción o pronóstico, donde se tiene el estado actual del sistema (posiblemente con error) y un mapeo de la progresión del sistema a medida que avanza el tiempo. En esta etapa se predice el siguiente posible estado del sistema. La segunda es la etapa de corrección, donde se combina el valor real medido con la predicción de la primera etapa para obtener un nuevo estado del sistema más exacto.

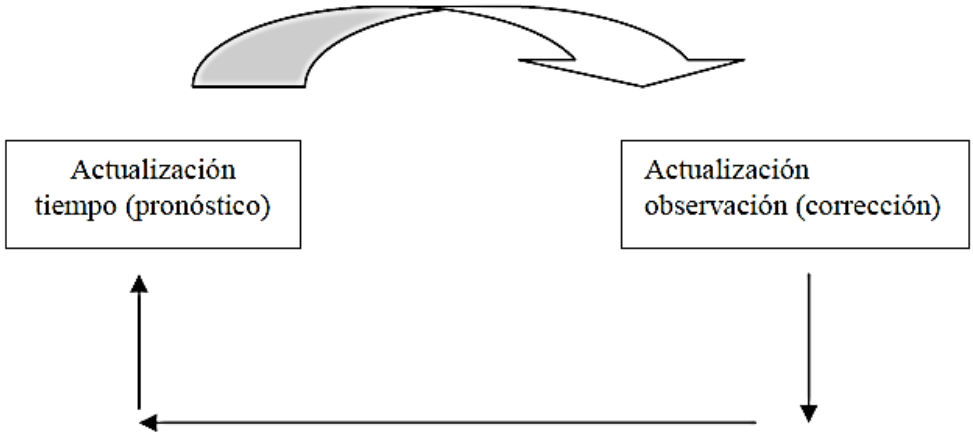

Figura 5. Ciclo del filtro de Kalman.

Lo que hace al filtro tan interesante es su habilidad para predecir el estado de un sistema en el pasado, presente y futuro, aún cuando la naturaleza precisa del sistema modelado es desconocida. En la práctica, las variables estado individuales de un sistema dinámico no pueden ser exactamente determinadas por una medición directa. Dado esto, su medición se realiza por medio de procesos estocásticos que involucran algún grado de incertidumbre en la medición.

El filtro de Kalman tiene como objetivo resolver el problema general de estimar el estado $X \in \Re^n$ de un proceso controlado en tiempo discreto, el cual es dominado por una ecuación lineal en diferencia estocástica de la siguiente forma:

$$X_k = AX_{k-1} + Bu_{k-1} + w_{k-1}$$

Con una medida $Z \in \Re^m$, que es:

$$Z_k = HX_k + v_k$$

Las variable w_k y v_k representan el error del proceso y de la medida respectivamente. Se asume que son independientes entre ellas, que son ruido blanco y con distribución de probabilidad normal:

$$p(w) \cong N(0, Q)$$

$$p(v) \cong N(0, R)$$

En la práctica, las matrices de covarianza de la perturbación del proceso, Q, y de la perturbación de la medida, R, podrían cambiar en el tiempo, pero por simplicidad se asumen constantes.

La matriz A se asume de una dimensión nxn y relaciona el estado en el periodo previo *k-1* con el estado actual *k*. La matriz H, de dimensiones mxn, relaciona el estado con la medición Z_k. Estas matrices pueden cambiar con el tiempo, pero también se asumen constantes por simplicidad.

Ya tenemos definido el proceso a ser estimado. Ahora debemos aplicar el algoritmo del filtro de Kalman. La primera etapa, de predicción, pronostica las estimaciones del estado y la covarianza.

$$\hat{X}_k^* = A\hat{X}_{k-1} + Bu_{k-1}$$

$$P_k^* = AP_{k-1}A^T + Q$$

Q representa la covarianza de la perturbación aleatoria del proceso que trata de estimar el estado.

En la segunda etapa, de corrección, encontramos la ganancia de Kalman K_k, seleccionada de tal forma que minimice la covarianza del error de la nueva estimación del estado. Luego se mide realmente el proceso para obtener Z_k y entonces generar una nueva estimación del estado $\hat{X}_k$, que incorpora la nueva observación. El paso final es obtener una nueva estimación de la covarianza del error. El algoritmo descrito para la segunda etapa es el siguiente:

$$K_k = P_k^* H^T (HP_k^* H^T + R)^{-1}$$

$$\hat{X}_k = \hat{X}_k^* + K_k\left(Z_k - H\hat{X}_k^*\right)$$

$$P_k = (I - K_k H)P_k^*$$

Después de cada par de actualizaciones, tanto del tiempo como de la medida, el proceso es repetido tomando como punto de partida las nuevas estimaciones del estado y de la covarianza del error. Esta naturaleza recursiva es una de las características llamativas del filtro de Kalman.

La Figura 6 ofrece un cuadro completo de la operación del filtro.

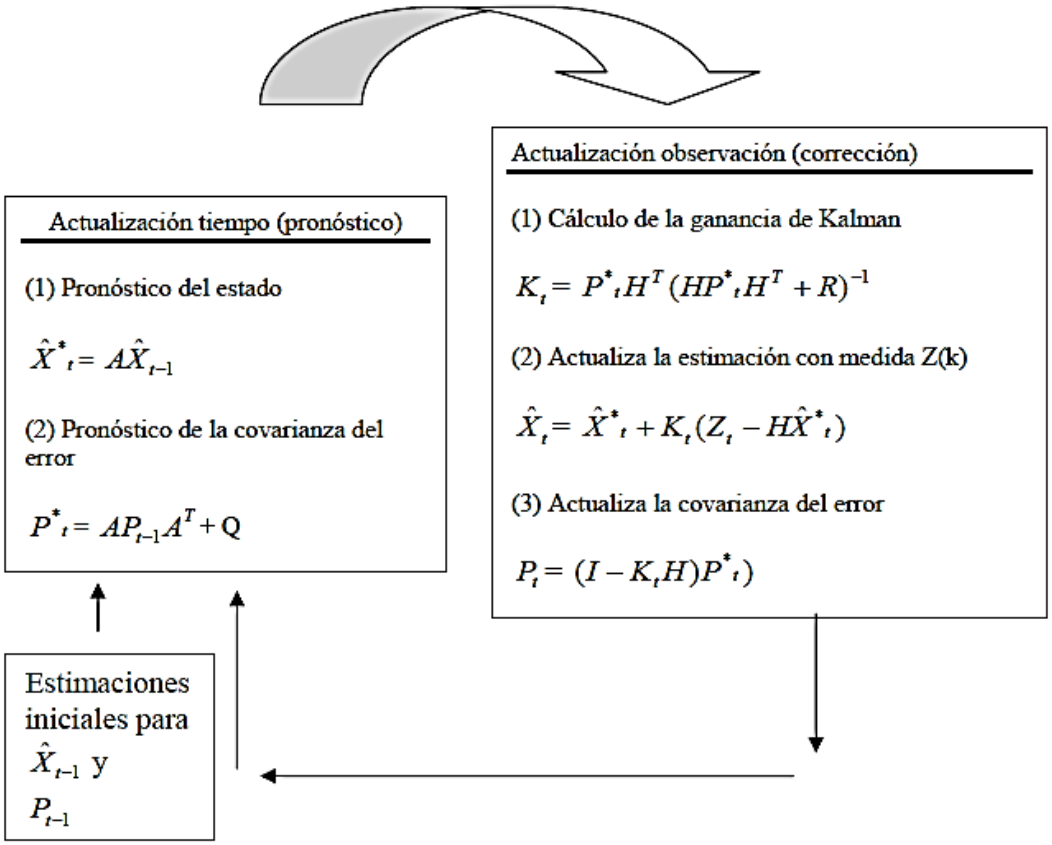

Figura 6. Visión completa del filtro de Kalman.

2.3 FPGA

Una FPGA es un circuito integrado de propósito general y lógica programable. Su configuración se especifica generalmente utilizando un lenguaje de descripción hardware o HDL, similar al utilizado para circuitos integrados de aplicación específica (ASIC). Una FPGA puede ser programada y usada para implementar cualquier función lógica que pueda implementar un ASIC, aunque se diferencian en que la FPGA se puede reprogramar si cambiamos la aplicación para la que está destinada.

La FPGA contiene bloques de lógica programable y una multitud de interconexiones reconfigurables que permite conectar los bloques entre sí, como nos muestra la Figura 7. Los bloques lógicos pueden ser configurados para llevar a cabo complejas funciones combinacionales o para implementar puertas lógicas como las conocidas AND o OR. La mayoría de FPGAs incluyen en los bloques lógicos elementos de memoria como simples flip-flops.

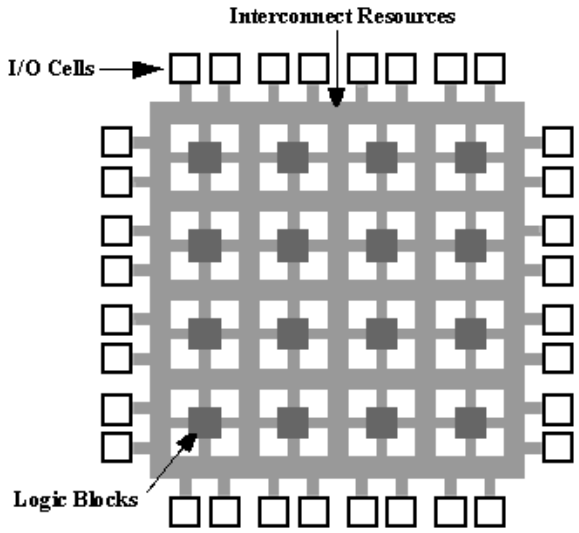

Figura 7. Arquitectura interna de una FPGA.

En la arquitectura de una FPGA también encontramos las celdas de entrada y salida, que nos permiten pasar al dispositivo los parámetros deseados obteniendo así la respuesta programada.

2.3.1. Historia de la FPGA

La industria de la FPGA surge del resultado de la convergencia de dos tecnologías diferentes, los dispositivos lógicos programables (PLDs) y los circuitos integrados de aplicación específica (ASIC). La historia de los PLDs comenzó con los primeros dispositivos PROM[8] y se les añadió versatilidad con los PAL[9], que permitieron un mayor número de entradas y la inclusión de registros. Mientras, los ASIC siempre han sido potentes dispositivos, pero su uso ha requerido una considerable inversión tanto de tiempo como de costes. Finalmente, combinando las dos estrategias con un mecanismo de interconexión programable mediante fusibles, antifusibles o celdas RAM, obteníamos la FPGA.

La primera FPGA viable comercialmente fue inventada en el año 1984 por Ross Freeman y Bernard Vonderschmitt, co-fundadores de Xilinx. Esta FPGA, modelo

[8] *Programmable Read-Only Memory* (Memoria Programable de Sólo Lectura).

[9] *Programmable Array Logic* (Matriz de Lógica Programable).

XC2064, constaba de puertas lógicas e interconexión entre éstas programables. Fue el comienzo de una nueva tecnología y de un nuevo mercado.

Hoy en día existen diferentes compañías que compiten en el mercado de las FPGAs como Xilinx, una de las más populares en grupos de investigación, Atmel, Altera, AMD y Motorola. Una tendencia reciente ha sido la de adoptar el enfoque de arquitectura un paso más allá mediante la combinación de los bloques lógicos e interconexiones de las FPGA con microprocesadores integrados y periféricos relacionados para formar un completo sistema en un chip programable. Ejemplos de tales tecnologías híbridas se pueden encontrar en los dispositivos de Xilinx Virtex-II Pro (Figura 8) y Virtex-4 que incluyen uno o más procesadores PowerPC.

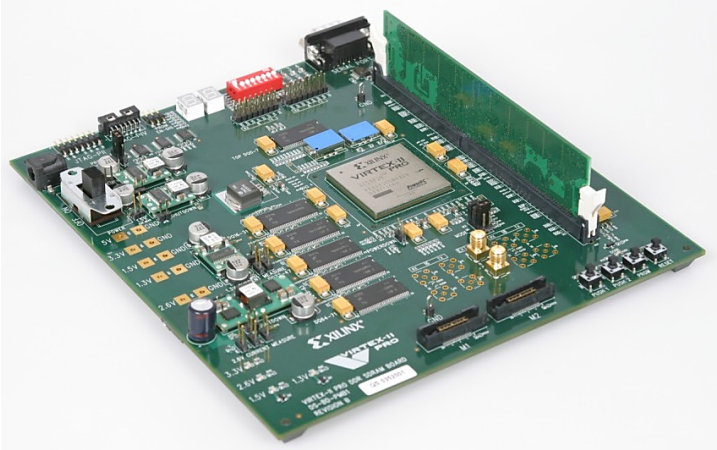

Figura 8. FPGA de Xilinx - Virtex-II Pro.

Muchas FPGA modernas soportan la reconfiguración parcial del sistema, permitiendo que una parte del diseño sea reprogramada mientras las demás partes siguen funcionando. Este es el principio de la idea de la computación reconfigurable.

2.3.2. Aplicaciones

Cualquier circuito de aplicación específica puede ser implementado en una FPGA, siempre y cuando ésta disponga de los recursos necesarios. Las aplicaciones donde más comúnmente se utilizan las FPGAs incluyen a los procesadores digital de señal (DSP), radio definido por software, sistemas aeroespaciales y de defensa, prototipos de ASICs, sistemas de imágenes para medicina, sistemas de visión para computadoras, reconocimiento de voz, bioinformática, entre otras.

Su uso en otras áreas es cada vez mayor, sobre todo en aquellas aplicaciones que requieren un alto grado de paralelismo. Un ejemplo de este tipo de aplicaciones es el desciframiento de códigos, en particular, el ataque de fuerza bruta de los códigos criptográficos. El paralelismo inherente de los recursos lógicos de una FPGA permite un rendimiento de procesamiento computacional considerable, incluso a velocidades de reloj de baja frecuencia.

En aplicaciones convencionales de alto rendimiento de computación también se está abriendo paso el uso de la FPGA, utilizando así el núcleo de la FPGA en lugar de un microprocesador.

Tradicionalmente, las FPGAs también tienen una gran utilidad para aplicaciones específicas en que el volumen de producción es pequeño. De esta manera, la prima que pagan las empresas en los costos por unidad de hardware de un chip programable es más asequible que los recursos de desarrollo dedicados a la creación de un ASIC para una aplicación de bajo volumen. Hoy en día, nuevos costes y la mejora del rendimiento de las FPGAs han ampliado la gama de aplicaciones viables para su uso.

2.3.3. Arquitectura

La mayoría de las arquitecturas de las FPGAs consiste en una matriz de bloques lógicos, celdas de entrada/salida y canales de conexión. Generalmente, todos los canales de interconexión tienen el mismo número de vías o uniones, e interconectan los bloques vertical y horizontalmente.

En general, un bloque lógico consiste en un pequeño número de celdas lógicas, típicamente formadas por 4 LUT (*Lookup Table*) de entrada, un sumador, tres multiplexores y un flip-flop tipo D como nos muestra la Figura 9. La salida puede ser síncrona o asíncrona, dependiendo de la programación del multiplexor de la derecha en la figura.

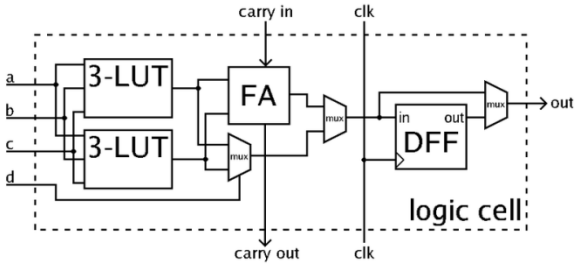

Figura 9. Ejemplo simple de una celda lógica.

2.3.4. Procesador PowerPC 405

El núcleo del PowerPC 405 es una implementación de 32 bits de un procesador RISC[10] PowerPC embebido. Está integrado en los dispositivos Xilinx Virtex-II Pro y Virtex-4, utilizando la tecnología de inmersión IP que soporta la infraestructura de bus CoreConnect[11] y extensas IP para periféricos y utilidades. Una gran variedad de aplicaciones utilizan esta arquitectura. Algunas de ellas son cámaras digitales, módems, teléfonos móviles y dispositivos GPS entre otros.

En la Figura 10 tenemos representada la arquitectura general de un sistema embebido con procesador PowerPC.

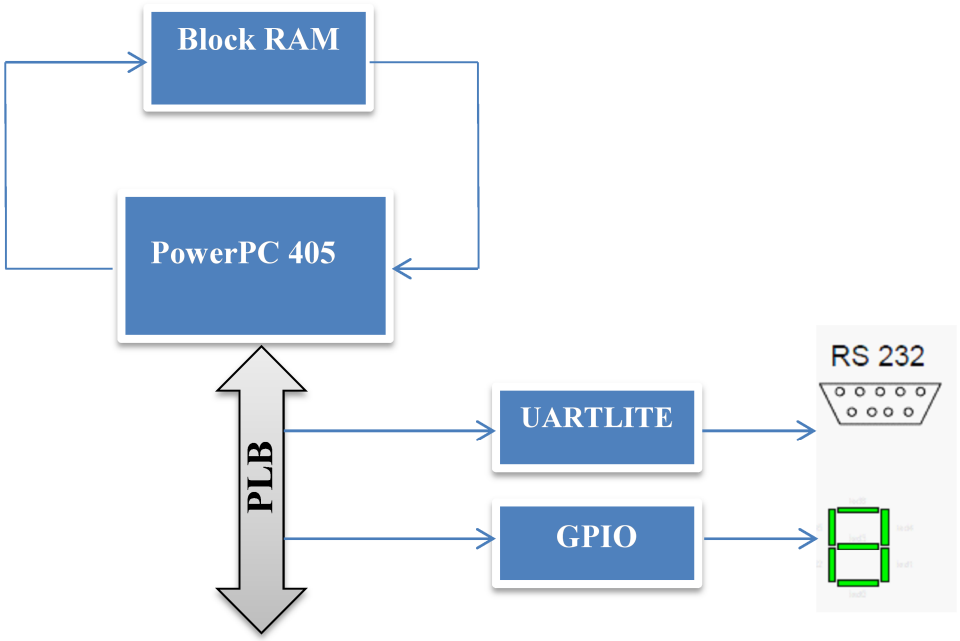

Figura 10. Arquitectura del sistema con PowerPC.

[10] *Reduced Instruction Set Computing* (Computadora con Conjunto de Instrucciones Reducidas).

[11] CoreConnect es una arquitectura de bus para microprocesadores desarrollada por IBM para Sistemas en Chip (SoC).

3. Descripción General del Proyecto

3.1 IMUs del Mercado

La primera parte del proyecto consiste en estudiar las diferentes unidades de medida inercial ofrecidas en el mercado actual y obtener una relación calidad/precio suficientemente buena para nuestro proyecto. Una vez estudiada gran parte del mercado y rechazado una serie de unidades debido a su elevado coste o sus bajas prestaciones, la elección de la IMU se encontraba reducida a dos unidades diferentes.

3.1.1. IMU 6DOF Razor - Ultra-Thin IMU

Esta pequeña unidad mostrada en la Figura 11 se compone por dos giróscopos y un acelerómetro. Los giróscopos son un LPR530AL para calcular el cabeceo (pitch) y alabeo (roll) y un LY530ALH para calcular la guiñada (yaw). El rango de estos giróscopos es de 300 º/s y unas sensibilidades de 0.83 y 3.33 mV/°/s dependiendo de la amplificación que le proporcionemos a la salida. El acelerómetro es un ADXL335 de tres ejes, con un rango de ± 3g y 300 mV/g de sensibilidad.

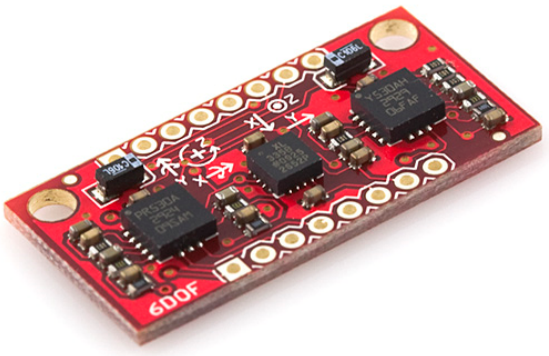

Figura 11. IMU 6DOF Razor.

Estos sensores hacen que la IMU tenga 6 grados de libertad, ya que calculan aceleración en los tres ejes y rotación en los tres ángulos de navegación.

El precio de esta unidad es de 75.18 €[12].

[12] Precio Marzo 2009. Éste puede variar con el tiempo.

3.1.2. Atomic IMU 6 Degrees of Freedom - XBee Ready

Esta unidad mostrada en la Figura 12 está compuesta por tres giróscopos LISY300AL de 300 º/s de rango y 3.3 mV/º/s de sensibilidad y por un acelerómetro MMA7260Q de tres ejes con opción a diferentes rangos y sensibilidades como nos muestra la Tabla 1:

Rango	Sensibilidad
1.5g	800 mV/g
2g	600 mV/g
4g	300 mV/g
6g	200 mV/g

Tabla 1. Rangos y sensibilidades para el acelerómetro MMA7260Q.

También incluye un microprocesador Atmel ATMega168 de 10 MHz de reloj con un conversor analógico-digital de seis canales de 10 bits para digitalizar las señales de los sensores.

Utiliza un protocolo de comunicación UART a 115200 baudios por segundo y viene preparada para poder utilizar el protocolo inalámbrico XBee. Aunque para ello se debe comprar un módulo XBee externo y añadirlo a la unidad.

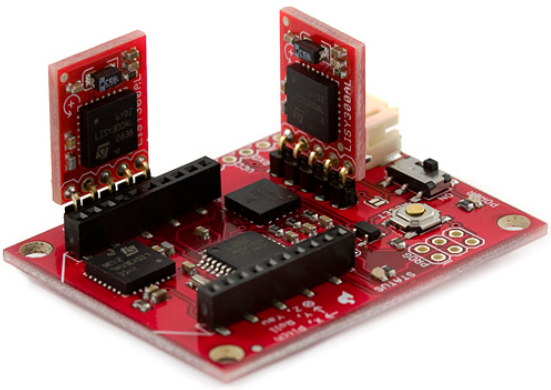

Figura 12. ***Atomic IMU 6DOF – XBee Ready*****.**

Esta unidad tiene un precio de 94.94 €[13].

[13] Precio Marzo 2009. Éste puede variar con el tiempo.

3.2 IMU Adquirida

En el apartado anterior hemos visto algunas características de las dos unidades de medida inercial con una relación calidad/precio asequible para este proyecto. Una vez analizadas, decidimos comprar la *Atomic IMU 6DOF – XBee Ready* debido principalmente a la mejora en los sensores y al microprocesador que ésta lleva integrado. Aunque el error que introducen los giróscopos es elevado respecto otras IMU menos económicas del mercado.

La *Atomic IMU* viene de fábrica con un programa en su microprocesador que nos permite, usando un terminal, cambiar la frecuencia de funcionamiento de la misma, activar o desactivar sensores según nos convenga y cambiar su sensibilidad, y escoger el tipo de salida entre binario o ASCII[14]. Esta configuración puede resultar interesante en según qué aplicaciones, pero para nuestro proyecto cambiaremos este programa para adaptarlo y optimizarlo a una mayor frecuencia de trabajo.

3.2.1. Comparación *Atomic IMU* con la unidad de la ETSE

Ahora compararemos la IMU que tenemos en la escuela con la nueva unidad que queremos adquirir. En la Tabla 2 podemos comparar los sensores de cada IMU y otras características de que se componen.

		Atomic IMU	**IMU ETSE**
Acelerómetros	Tipo	MMA7260Q	ADXL202
	Rango de Medida	±1.5g/2g /4g/6g	±2g
	Ancho de Banda (f_{-3dB})	Eje x, y: 300 Hz Eje z: 150 Hz	500 Hz
	Densidad de Ruido	350 µ*g*/µ*Hz*	500 µ*g*/µ*Hz*
	Precio (unidad)	8.97 €	7.27 € (1000 uds)

[14] *American Standard Code for Information Interchange* (Código Estadounidense Estándar para el Intercambio de Información).

Giróscopos	Tipo	LISY300AL	ENC-03JA
	Rango de Medida	300 º/seg	300 º/seg
	Sensibilidad	3.3 mV/º/seg	0.67 mV/º/seg
	Respuesta frec.	88 Hz	50 Hz
	Precio (unidad)	9.08 €	9.63 € (100 unidades)
Otros		ATMega168TM	Conversor A/D (3.17€)
		Batería LiPo	
	Precio IMU	94.94 €	≥ 53.87 € (solo sensores)

Tabla 2. Comparación entre *Atomic IMU* y la unidad de la ETSE.

En el precio de la IMU de la Universidad sólo se ha considerado el precio aproximado de los sensores. Deberíamos considerar también el precio de la fabricación y de otros posibles componentes como condensadores, resistencias, etc. También tenemos que tener en cuenta que la *Atomic IMU* tiene un procesador integrado que nos ayudará a la hora de enviar los datos hacia la FPGA.

Así pues, exceptuando el ancho de banda de los acelerómetros, podemos decir que la *Atomic IMU* tiene mejores prestaciones que la IMU de la Universidad en cuanto a sensibilidad de los sensores, ruido, precio y tipo de la señal de salida.

3.3 Funcionamiento y Programación de la Atomic IMU

Como hemos comentado en la sección 3.2, el microprocesador ATMega168 que incorpora la *Atomic IMU* viene con un programa establecido por defecto[15]. En este

[15] El programa por defecto de la *Atomic IMU* se puede descargar de la siguiente página de *Sparkfun*: http://www.sparkfun.com/commerce/product_info.php?products_id=9184

apartado veremos el funcionamiento de la IMU así como la modificación del programa para adaptarlo a los requisitos del proyecto. También estudiaremos los diagramas de tiempo para las muestras de los sensores. Estos diagramas los utilizaremos para determinar aproximadamente la frecuencia máxima de trabajo de la IMU a la que podemos aspirar.

El programa proporcionado por *Sparkfun* tiene a la unidad en estado de reposo inicialmente. Para hacer funcionar la IMU debemos cambiar su estado a través de un terminal (cómo por ejemplo el *HyperTerminal* de Windows). Para ello debemos conectar la IMU con el PC. Como el protocolo de comunicación de la unidad de medida inercial es UART a 3.3V y el del PC es RS-232 a 5V, debemos utilizar un conversor TTL/RS-232 como el MAX232. El circuito acondicionador y el conversor lo podemos ver en la Figura 13.

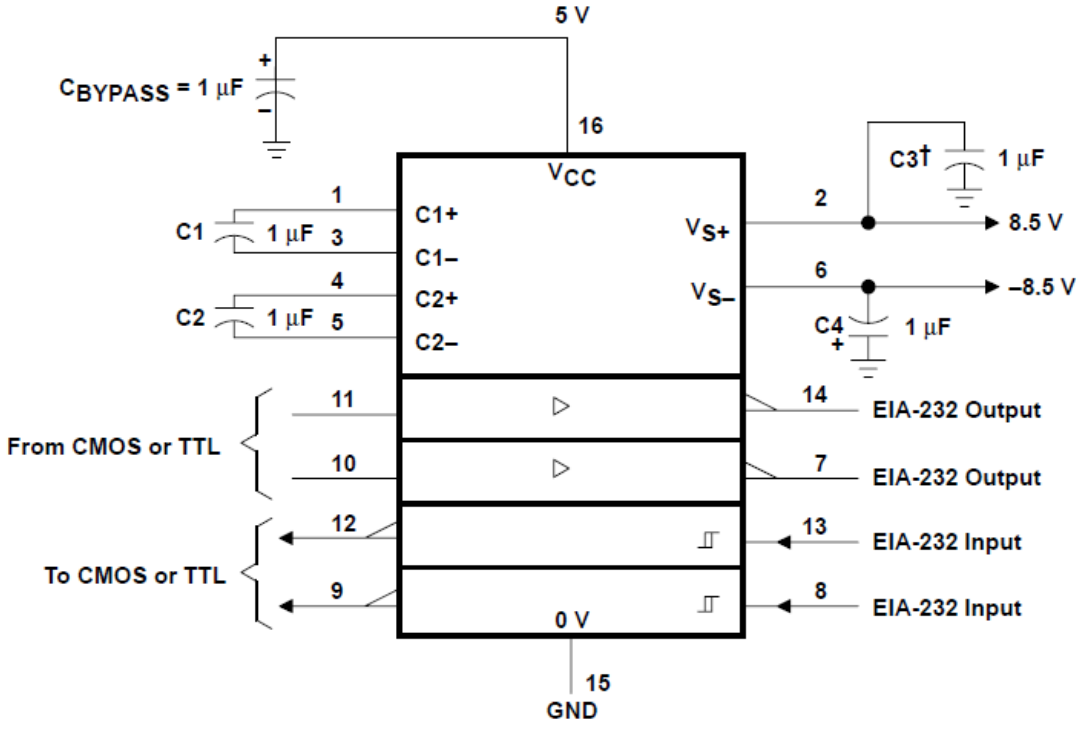

Figura 13. Conversor MAX232.

En la Figura 14 vemos el mismo circuito en la realidad, montado y preparado para conectar directamente la IMU. El PC lo conectamos mediante el cable visto en la Figura 15 al puerto serie.

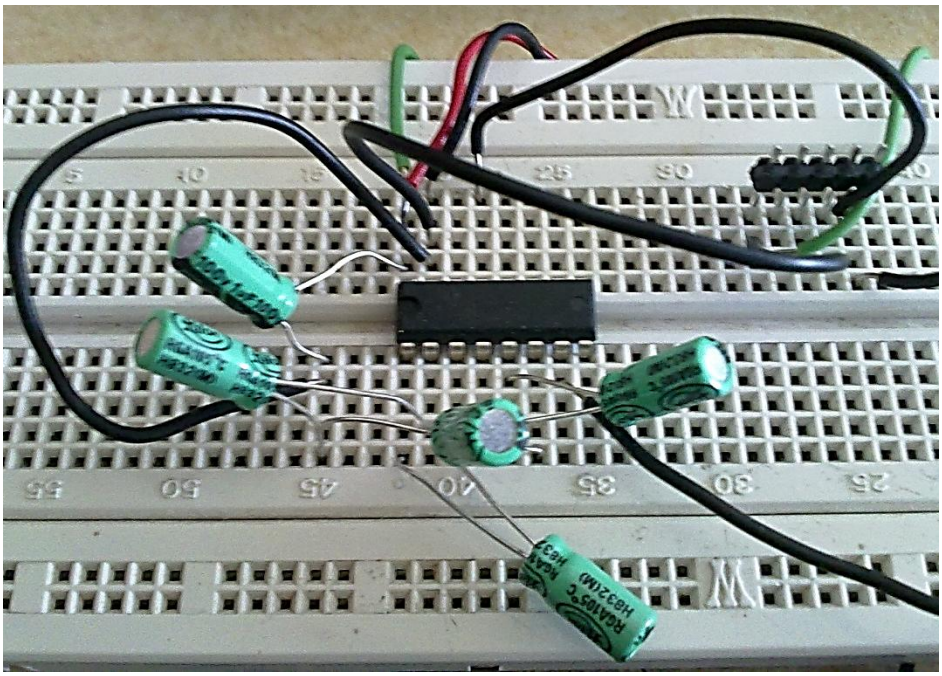

Figura 14. Conversor MAX232 y el circuito acondicionador montado.

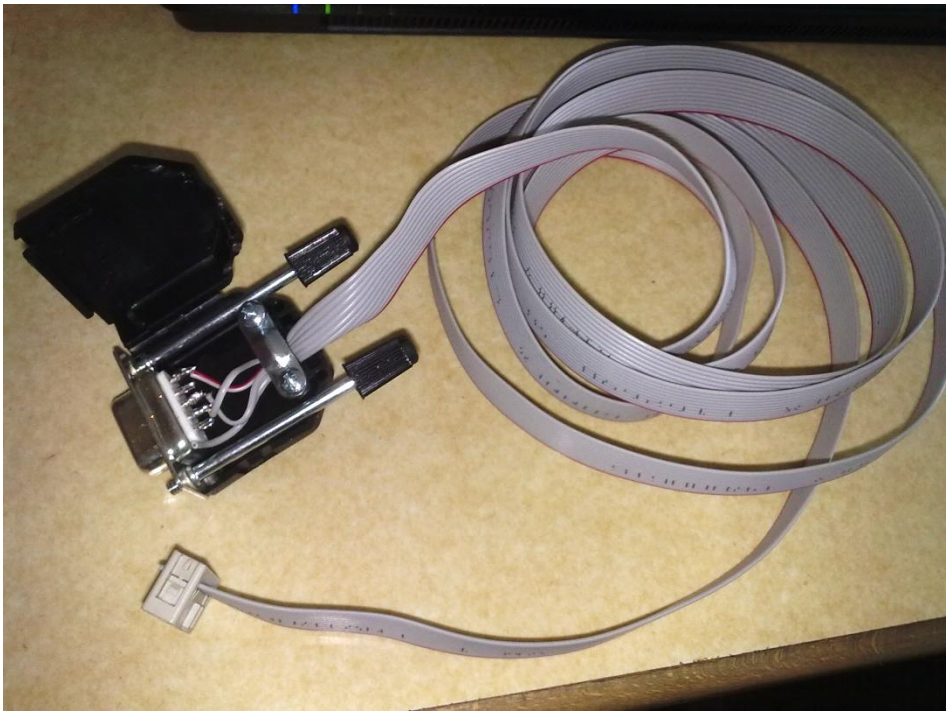

Figura 15. Cable para conectar el MAX232 con el PC mediante RS-232.

Cuando ya tenemos conectadas la IMU y el PC debemos configurar la conexión del terminal a 115200/8/N/1, lo que significa una velocidad de transmisión de 115200 baudios, con 8 bits de datos y uno de parada. Así es como está configurada la transmisión de la IMU.

Ahora, pulsando la barra espaciadora, ya podemos ver en el terminal el menú de configuración de la IMU. Si ahora activamos todos los sensores a una frecuencia de funcionamiento determinada, la unidad empezará a mandar tramas al PC. El formato de éstas es:

A acc_x acc_y acc_z pitch roll yaw Z

Dónde A y Z son caracteres de principio y final de trama. Las aceleraciones lineales y angulares se pueden representar en binario o en ASCII. Así pues, vemos que el formato de la trama incluye una cabecera y final, unas tabulaciones entre cada carácter y los valores de los sensores. Esto implica una pérdida del ancho de banda, ya que enviamos caracteres que no necesitamos como el tabulador.

La modificación propuesta para optimizar el programa es anular todos los caracteres que no nos aportan información, como los tabuladores e incluso el carácter de final de trama. También adaptaremos la configuración de la frecuencia y sensibilidad del acelerómetro para la FPGA y así poder ofrecer al usuario una cómoda y rápida configuración a través de los *switch* de la placa.

Para poder programar el microprocesador de la IMU necesitamos de un programa que descarga el código en un lenguaje de bajo nivel a la IMU mediante un cable programador como el que nos muestra la Figura 16. El programa utilizado es el Pony-Prog.

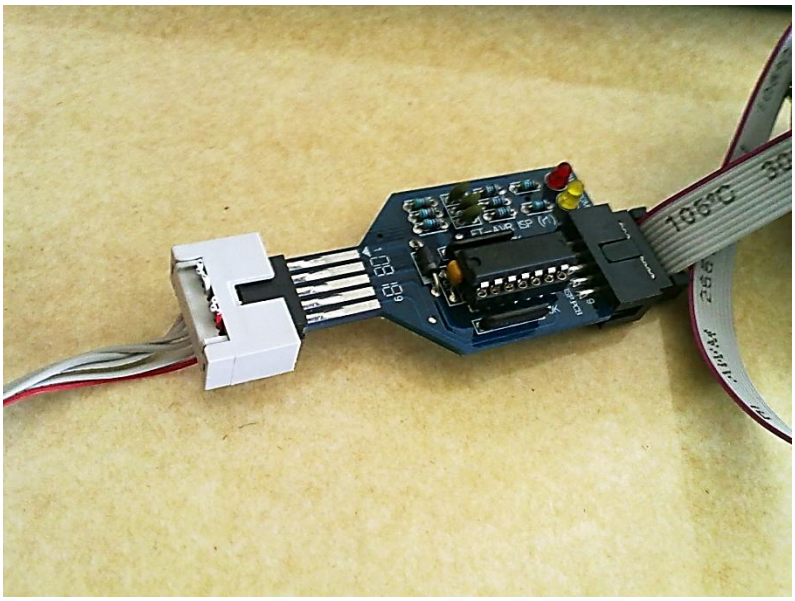

Figura 16. Programador Pony-Prog utilizado.

Las partes de la *Atomic IMU* comentadas hasta ahora para programar, para la conexión con el conversor MAX232 (el mismo que para la FPGA), el microprocesador y la alimentación de la unidad las podemos ver en la Figura 17.

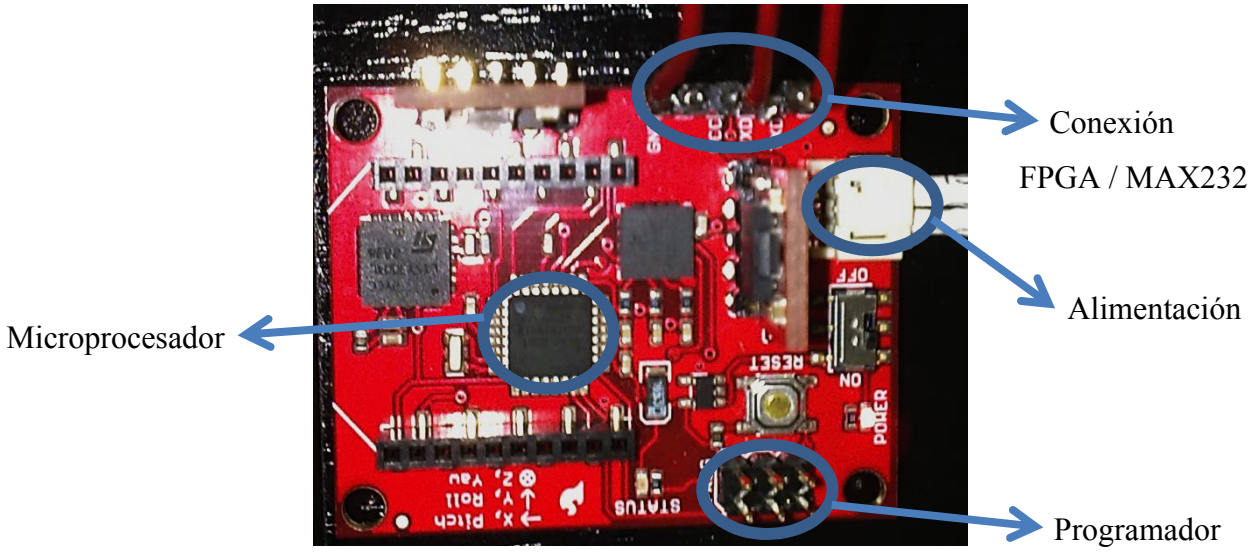

Figura 17. Vista superior de la *Atomic IMU*.

3.3.1. Diagramas de Tiempo

Para representar los diagramas de tiempo debemos calcular el tiempo de conversión del conversor analógico-digital teniendo en cuenta la frecuencia de reloj establecida. Por defecto, el microprocesador ATMega168 tiene un oscilador RC interno de 8 MHz y un preescaler de 8[16], con lo que la frecuencia de reloj por defecto es de 1 MHz. Al ADC le asignamos un preescaler de 64, con lo que podemos calcular el tiempo de reloj ADC como:

$$f_{ADC} = \frac{f_{CLK}}{Preescaler_{ADC}} = \frac{1\,MHz}{64} = 15.625\,kHz \;\rightarrow\; t_{ADC} = \frac{1}{f_{ADC}} = 64\,\mu s$$

La primera conversión del ADC dura 25 ciclos de reloj ADC, con lo que la conversión de un dato de sensor es:

$$t_{conv} = 64\,\mu s \cdot 25\,ciclos = 1.6\,ms$$

[16] Según el *datasheet* del ATMega168 (CKDIV8).

Entonces, para convertir todos los datos de una trama (un dato de cada sensor) necesitamos $6 \cdot t_{conv} = 9.6\,ms$. Este tiempo implica una frecuencia de trabajo de la unidad máxima de 100 Hz aproximadamente.

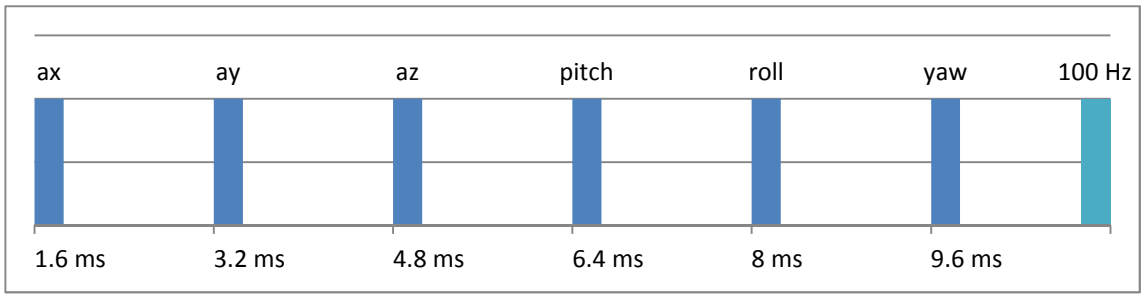

Figura 18. Diagrama de tiempo para una frecuencia de reloj de 1 MHz y una frecuencia de trabajo de 100 Hz.

Para conseguir una mayor frecuencia de trabajo, del orden de 200 – 300 Hz, debemos utilizar el reloj externo de 10 MHz que lleva incorporado el ATMega168 y quitar el preescaler de 8 para trabajar a una frecuencia de 10 MHz. En este caso, el tiempo de reloj del conversor ADC será:

$$f_{ADC} = \frac{f_{CLK}}{Preescaler_{ADC}} = \frac{10\ MHz}{64} = 156.25\ kHz \ \rightarrow \ t_{ADC} = \frac{1}{f_{ADC}} = 6.4\ \mu s$$

Y la duración de una conversión:

$$t_{conv} = 6.4\ \mu s \cdot 25\ ciclos = 160\ \mu s$$

El tiempo de conversión de toda una trama mejora a $6 \cdot t_{conv} = 960\ \mu s$, con lo que podemos aplicar una frecuencia máxima de 1 kHz teóricamente.

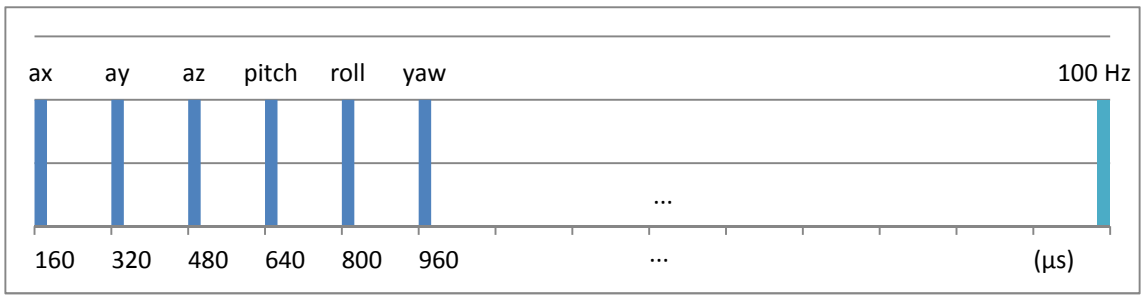

Figura 19. Diagrama de tiempo para una frecuencia de reloj de 10 MHz y una frecuencia de trabajo de 100 Hz.

En la práctica, estos tiempos varían ya que el microprocesador no sólo captura datos, sino que tiene instrucciones del tipo *if* y del tipo *while*, entre otras, que consumen

muchos ciclos de reloj y que no estamos teniendo en cuenta. Además, también debemos considerar la velocidad de transmisión y recepción, ya que, por mucho que aumentemos la frecuencia de trabajo de la unidad, sólo podremos recibir como máximo a la frecuencia de transmisión de 115200 baudios.

3.4 Datos obtenidos de la IMU y cálculo del ángulo. Filtro de Kalman

En este apartado describiremos los pasos necesarios para obtener la aceleración y la velocidad angular a partir de los datos de los sensores. También veremos como extraer los ángulos de dichos datos y la importancia de aplicar el filtro de Kalman para corregir el error de los giróscopos.

3.4.1. Aceleración

Para obtener la aceleración a partir de los datos del acelerómetro (a_{sensor}) debemos aplicar la siguiente ecuación:

$$a = \frac{(a_{sensor} - bias) \cdot V_{ref}}{Sens \cdot Res_{ADC}} \cdot g \left[{}^{m}/_{s^2} \right]$$

El bias lo obtenemos a partir de un estudio de la unidad en la que la mantenemos estática y capturamos datos. Una vez tenemos suficientes, hacemos la media de los datos de la aceleración en x, y, z. Esta media obtenida es el bias de la aceleración para cada coordenada. El V_{ref} (voltaje de referencia) y Sens (sensibilidad) lo extraemos del *datasheet*. La sensibilidad es variable, y puede ser 1.5g, 2g, 4g o 6g. La resolución del conversor analógico-digital es 1024, ya que convierte 10 bits y la g considera la gravedad.

La ecuación anterior simplemente escala el dato del sensor para obtener la aceleración correctamente. En la Figura 20 tenemos representados los datos obtenidos directamente del acelerómetro y la aceleración calculada de dichos datos. Como podemos ver la gráfica es la misma pero con otra escala. También podemos deducir que el acelerómetro es muy sensible a vibraciones o movimientos bruscos por los picos que se aprecian.

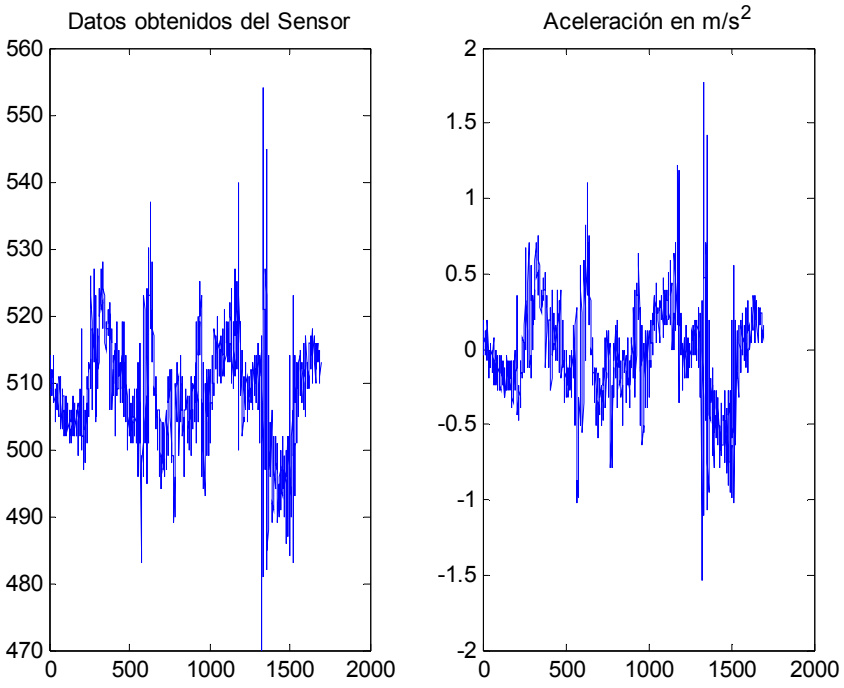

Figura 20. Datos obtenidos del acelerómetro y aceleración (en el eje x) para f = 150 Hz.

3.4.2. Ángulo a partir de la aceleración

En esta sección veremos como calcular el ángulo pitch y roll a partir de la aceleración obtenida con los acelerómetros, vista en el apartado anterior.

Sabemos que la gravedad es una aceleración, con lo que la podemos medir con nuestros acelerómetros. La Figura 21 nos muestra cómo podemos relacionar la gravedad y nuestra medida de aceleración con el ángulo θ y pitch respectivamente.

$$pitch = \theta + 90º$$

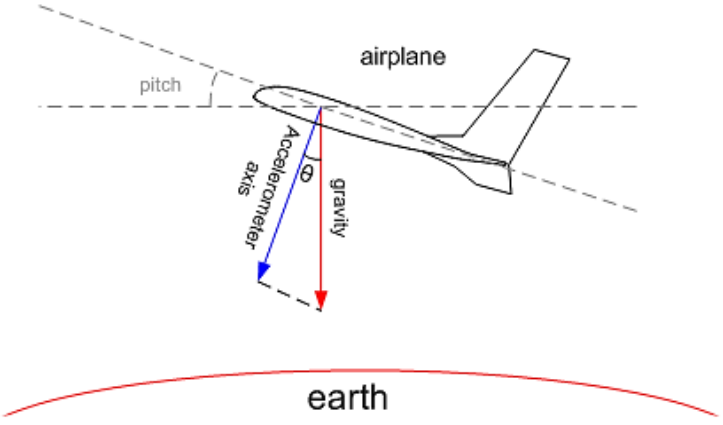

Figura 21. Ángulo pitch a partir del acelerómetro y la gravedad.

Por trigonometría podemos deducir la aceleración en el eje azul de la figura a partir del ángulo θ cómo:

$$aceleración = \cos\theta \cdot g \left[{}^{m}/_{s^2}\right]$$

Entonces, el ángulo será:

$$\theta = \cos^{-1}\left(\frac{aceleración}{g}\right)[º]$$

$$pitch = \sin^{-1}\left(\frac{aceleración}{g}\right)[º]$$

El inconveniente de esta sencilla solución es que el inverso del seno no nos proporciona 360 grados completos del ángulo pitch. Por ello necesitamos otro acelerómetro y aplicar la tangente de la siguiente forma:

$$pitch = -\tan^{-1}\left(\frac{a_y}{\sqrt{a_x^2 + a_z^2}}\right)[º]$$

Donde a_y, a_x y a_z son las aceleraciones en el eje y, eje x y eje z respectivamente.

Si calculamos este ángulo para la aceleración obtenida en la Figura 20, obtenemos lo mostrado en la Figura 22, en la cual vemos un movimiento de ± 90º. Aquí podemos comprobar como realmente el acelerómetro es muy sensible a las vibraciones, y al obtener el ángulo nos encontramos unos valores muy ruidosos.

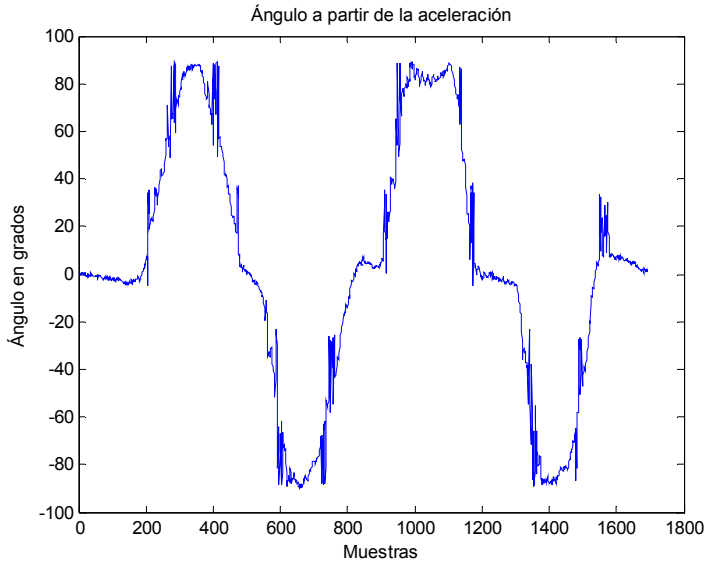

Figura 22. Ángulo pitch obtenido a partir de la aceleración para f = 150 Hz.

Para el caso del roll la ecuación que utilizamos es la siguiente:

$$roll = -\tan^{-1}\left(\frac{a_x}{\sqrt{a_y^2 + a_z^2}}\right)[º]$$

3.4.3. Velocidad Angular

Para los giróscopos tenemos una ecuación que nos extrae la velocidad angular a partir de los datos proporcionados. Esta ecuación contempla la sensibilidad del giróscopo, que para el LISY300AL vale 3.3 mV/º/s. Entonces obtenemos la velocidad angular como:

$$w = \left(\frac{w_{sensor} \cdot V_{ref}}{Res_{ADC}} - \frac{V_{ref}}{2}\right) \cdot \frac{1}{Sens}\left[{}^{º}/_{s^2}\right]$$

Como pasaba con la aceleración, la ecuación anterior sólo escala el dato del sensor para obtener velocidad angular. En la Figura 23 tenemos representados los datos del sensor y la velocidad angular de dichos datos.

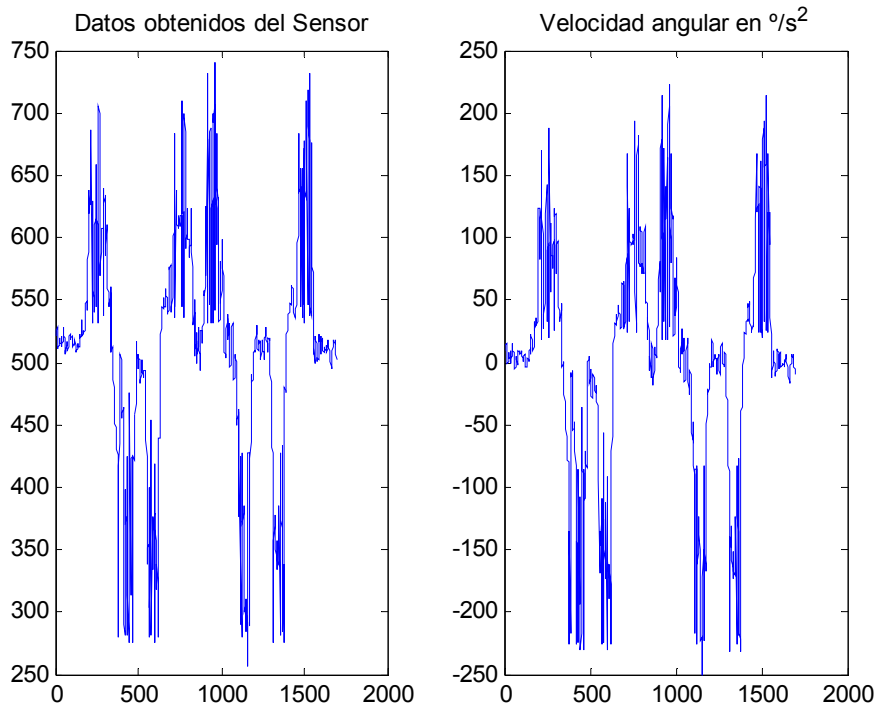

Figura 23. Datos obtenidos del sensor y velocidad angular (pitch) para f = 150 Hz.

3.4.4. Ángulo a partir de la velocidad angular

Lo que realmente nos interesa de la unidad es el movimiento que sufre, y en ello se incluye la rotación o el giro en grados. Para obtener esta rotación debemos integrar la

velocidad angular vista anteriormente. La forma de integrar que utilizaremos es la siguiente, en la que tenemos en cuenta muestras anteriores:

$$\alpha_i = \alpha_{i-1} + w_i \ [º]$$

Si integramos la velocidad angular del apartado anterior (Figura 23), obtenemos el ángulo pitch que tenemos representado en la Figura 24. En el segundo gráfico podemos apreciar el error de bias del giróscopo, ya que el movimiento causado en la IMU es de ± 90º y el ángulo encontrado tiende a aumentar con el tiempo. Para corregir este error necesitamos el filtro de Kalman, como veremos en el apartado 3.4.5.

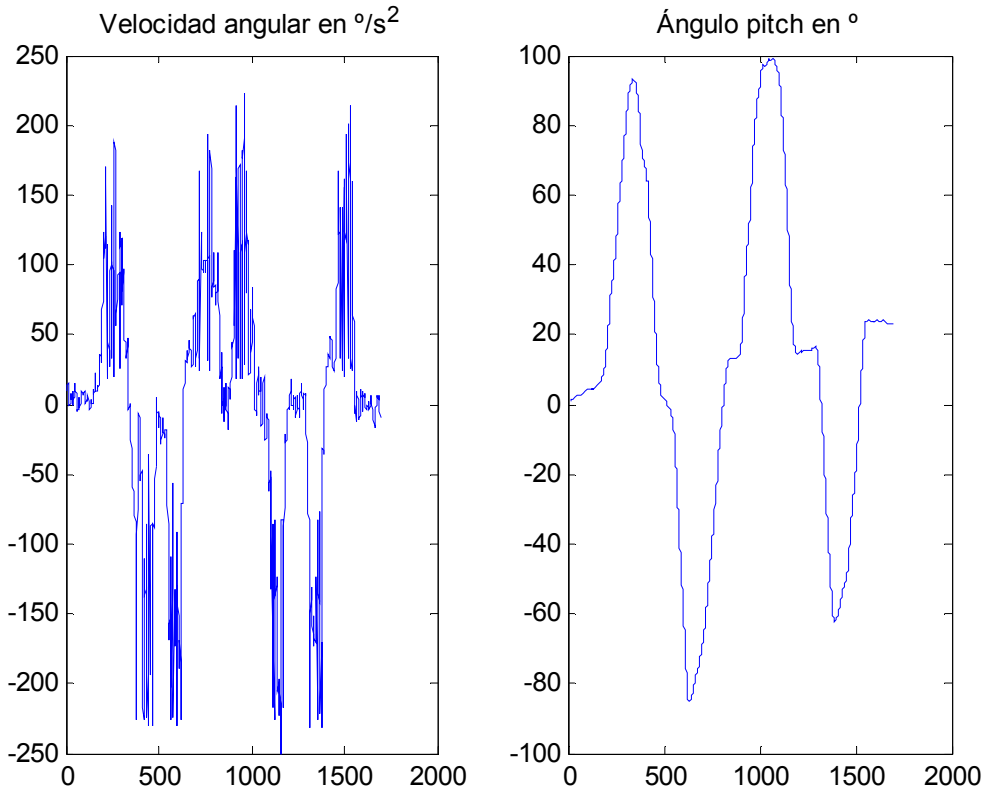

Figura 24. Velocidad angular y ángulo pitch obtenidos a partir de los datos del giróscopo para f = 150 Hz.

3.4.5. Filtro de Kalman. Resultados

Ahora debemos aplicar el filtro de Kalman visto en la sección 2.2. Para ello consideraremos los siguientes parámetros, que representan a nuestra unidad de medida inercial sobre el filtro:

- Matriz de estado $X = \begin{pmatrix} ángulo \\ bias \end{pmatrix}$
- Matriz de covarianza de la medida $R = 0.3$
- Matriz de covarianza del proceso $Q = \begin{pmatrix} 0.001 & 0 \\ 0 & 0.003 \end{pmatrix}$
- $A = \begin{pmatrix} 1 & -dt \\ 0 & 1 \end{pmatrix}$, con *dt* el tiempo entre medidas.
- $B = \begin{pmatrix} dt \\ 0 \end{pmatrix}$
- $H = (1 \quad 0)$
- El valor de Z_k es el ángulo obtenido con el acelerómetro como hemos visto en el apartado 3.4.2.
- El valor de u_k es el ángulo obtenido con el giróscopo, visto en la sección 3.4.4.

Si aplicamos el filtro con estos parámetros para el movimiento visto en la Figura 24 obtenemos una corrección del movimiento y del ángulo. Lo podemos observar en la Figura 25.

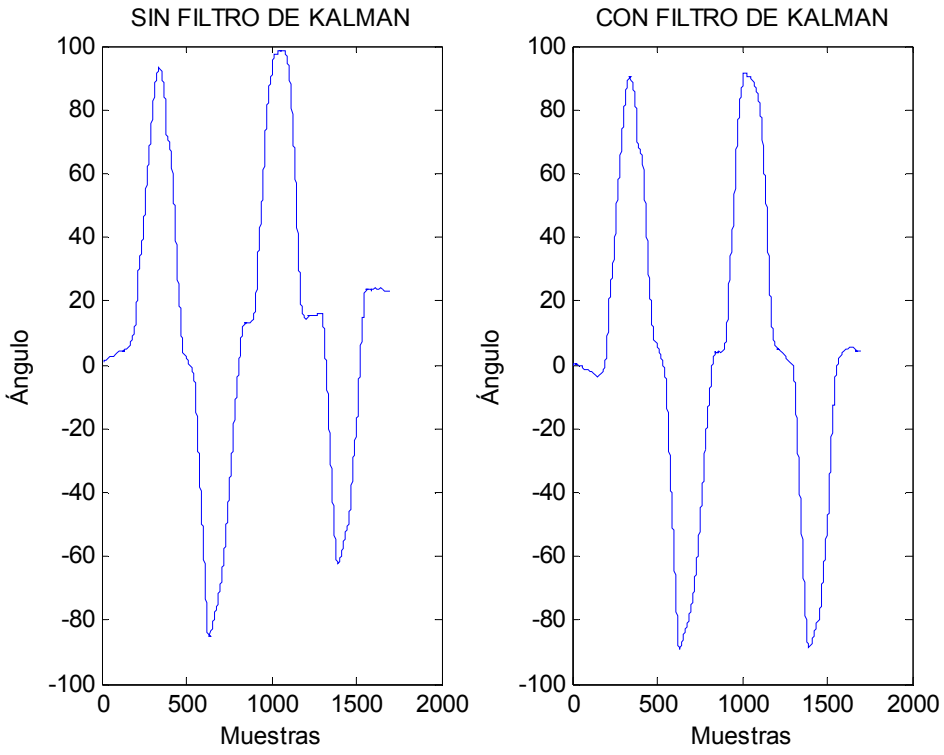

Figura 25. Ángulo pitch sin filtro y con filtro de Kalman para f = 150 Hz.

En la figura anterior podemos apreciar el error de bias del giróscopo sin utilizar el filtro. En cambio, al aplicar el filtro de Kalman, eliminamos este bias y obtenemos el ángulo exacto realizado en la IMU.

Otros ejemplos más del buen funcionamiento del filtro para obtener el ángulo pitch a diferentes frecuencias de trabajo lo vemos en la Figura 26, Figura 27 y Figura 28. En estos ejemplos se puede apreciar más el error de bias con el tiempo. Si no aplicáramos el filtro, en unos pocos segundos tendríamos un error de alguna centena de grados.

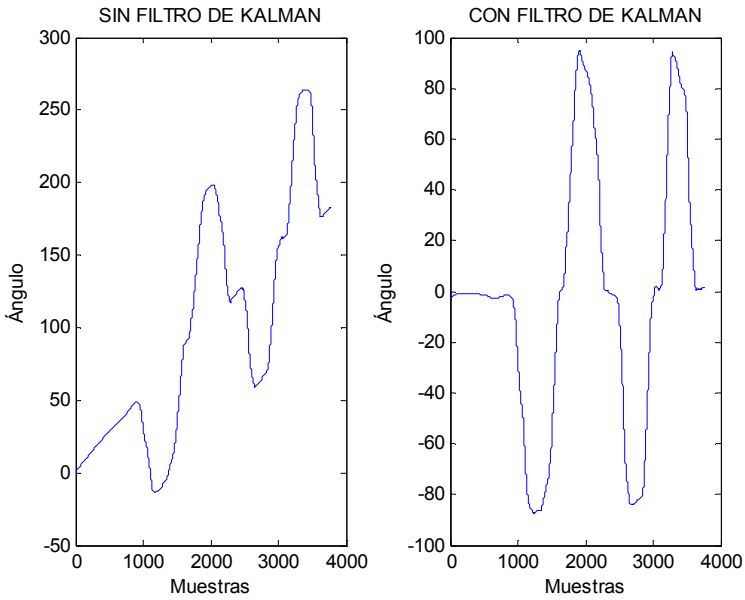

Figura 26. Ángulo pitch sin filtro y con filtro de Kalman para f = 150 Hz.

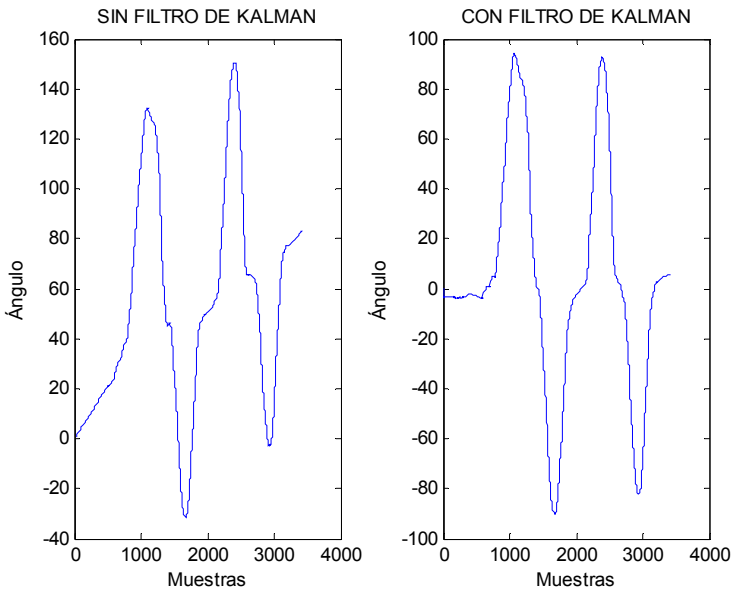

Figura 27. Ángulo pitch sin filtro y con filtro de Kalman para f = 200 Hz.

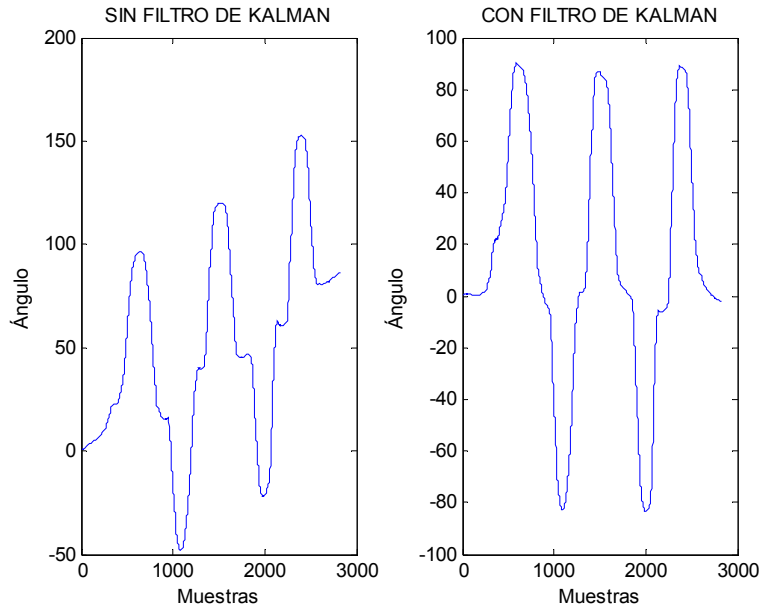

Figura 28. Ángulo pitch sin filtro y con filtro de Kalman para f = 250 Hz.

También aplicamos lo mismo para obtener el ángulo roll. Un ejemplo nos lo muestra la Figura 29. En este caso, a diferencia del anterior, el bias decrece en lugar de aumentar. En 4000 muestras tenemos un error de $320º - 90º = 230º$. Eso, a una frecuencia de 200 Hz, significa que en $4000 \cdot {}^{1}/_{200} = 20$ segundos tenemos un error de 230º. Con el filtro de Kalman eliminamos este error y obtenemos el ángulo roll deseado de ± 90º.

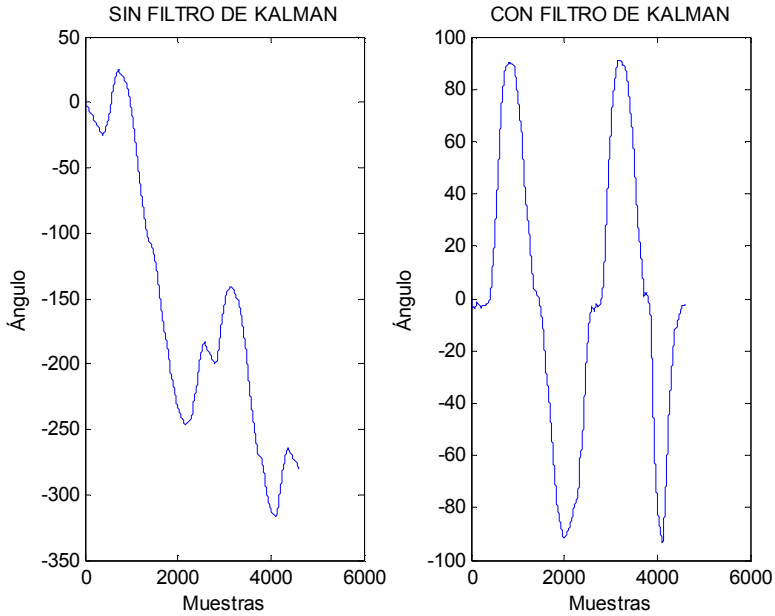

Figura 29. Ángulo roll sin filtro y con filtro de Kalman para f = 200 Hz.

El programa utilizado para obtener las figuras es Matlab y el código (obtención de los datos guardados en un fichero, cálculo de la aceleración y velocidad angular y aplicación del filtro de Kalman) se encuentra en el Anexo C.

3.5 Programación FPGA

Para la programación de la FPGA utilizamos un software específico de Xilinx llamado *Xilinx Platform Studio* (EDK). Esta potente herramienta nos permite ver el diagrama de bloques de la FPGA y crear o modificar las conexiones según los requisitos deseados.

Para nuestro sistema necesitamos una conexión UART para conectar directamente la IMU a la FPGA y una conexión RS-232 para conectar el PC a la FPGA y poder observar su funcionamiento. Estas conexiones se manejan mediante el controlador UARTLITE. También necesitamos *switches* y pulsadores, que en este caso se manejan mediante el controlador GPIO.

Al crear un nuevo proyecto con el software EDK, observamos que sólo hay un driver UART definido, el RS-232. Entonces, debemos definir el controlador UART para el periférico, en este caso la IMU. Para añadir el periférico lo buscamos en el programa mediante el *IP Catalog*, lo conectamos al bus PLB y lo configuramos para las condiciones de la IMU (115200 baudios, 8 bits de datos, sin paridad). Una vez hecho esto, debemos decirle al procesador por qué puerto UART conectaremos el periférico. En el *datasheet* de la FPGA VirtexII-Pro encontramos el nombre de todos los puertos de entrada y salida, donde escogemos los puertos UART L4 como receptor y N5 como transmisor (con nivel de voltaje TTL).

Para el caso de los *switches* y de los pulsadores, observamos que ya tienen definido cada uno sus controladores. En la Figura 30 vemos el diagrama de bloques de la conexión PLB, donde tenemos representados los *switches* en A, los pulsadores en C, el controlador RS-232 en D y el periférico añadido por nosotros para conectar la IMU llamado xps_uartlite_0.

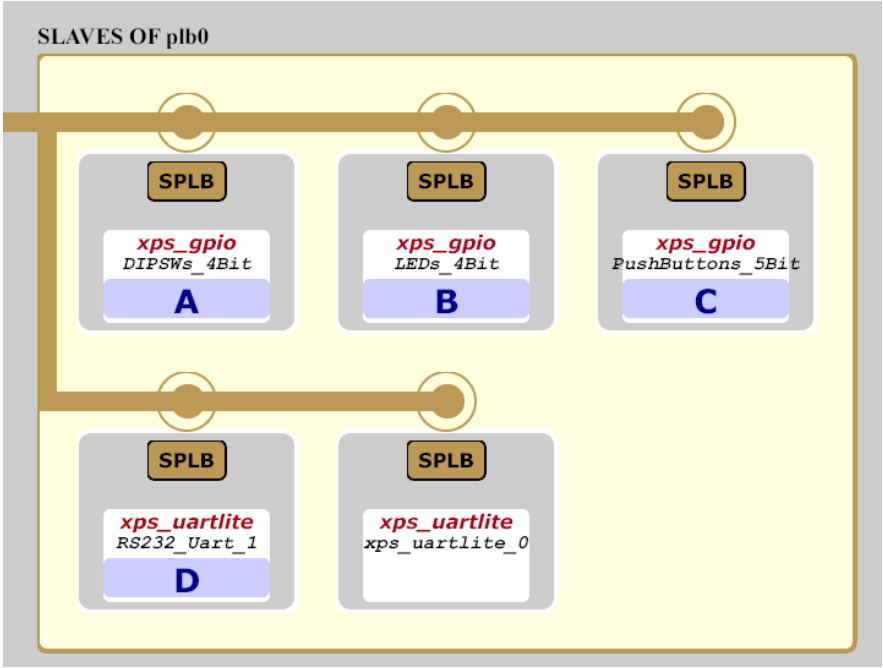

Figura 30. Diagrama de bloques de la conexión PLB de la FPGA con los controladores GPIO y UARTLITE.

El programa diseñado para recoger los datos de la IMU y poder configurar la misma mediante los pulsadores y los *switches* lo podemos encontrar en el Anexo D. Básicamente, consiste en un bucle infinito que lee datos de la IMU mientras el receptor no esté vacío. En el caso que pulsemos el pulsador, enviará a la unidad una trama consistente en un 1 que indica que hemos pulsado y 4 bits que indican el estado de los *switches*. El programa de la IMU está diseñado para identificar si se ha pulsado el pulsador, lo que indica que queremos configurarla, y extraer la información de frecuencia y sensibilidad de los 4 bits. La Tabla 3 nos muestra la configuración que interpreta la IMU para los 4 bits que envía la FPGA.

Bits más significativos	**Sensibilidad (g)**	**Bits menos significativos**	**Frecuencia (Hz)**
00	1.5	00	100
01	2	01	200
10	4	10	250
11	6	11	300

Tabla 3. Valor de los 4 bits que se configuran mediante los *switches*.

3.6 Programa en Visual Basic

Como ampliación del trabajo y para una mayor visualización del funcionamiento de la IMU, hemos modificado un programa en Visual Basic proporcionado por *Sparkfun*[17] llamado *Atomic Mixer*, en el cual tenemos representadas siete barras que se llenan o vacían según el valor de cada sensor (exceptuando la primera que es el contador). Cada barra representa los datos de un sensor en ASCII, es decir, tal y cómo provienen de la IMU. Entonces, éstos solo nos dan una referencia del movimiento que aplicamos a la IMU según se llenan las barras. En esta aplicación nos permiten escoger la frecuencia de trabajo y la sensibilidad de los acelerómetros, tal y como podemos comprobar en la Figura 31.

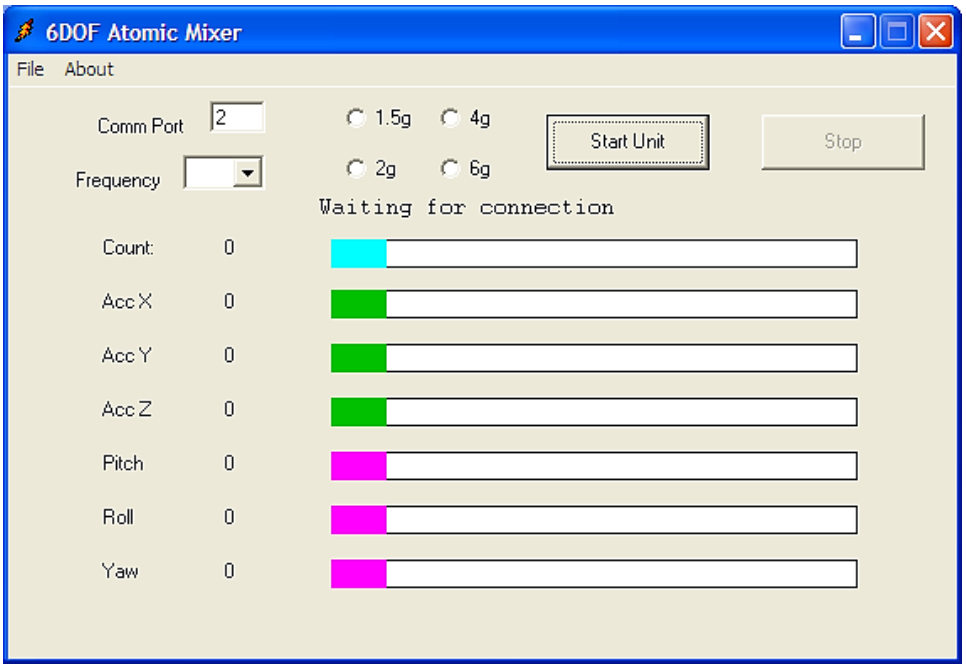

Figura 31. ***Atomic Mixer*** **proporcionado por** ***Sparkfun*****.**

Nuestra mejora al programa consiste en añadir una figura que se mueva según el usuario mueve la IMU en tiempo real. Para ello debemos calcular el ángulo de los giróscopos (sección 3.4.4), el de los acelerómetros (sección 3.4.2) y aplicar el filtro de Kalman visto en el apartado 3.4.5, ya que si aplicamos directamente los datos del giróscopo sin el filtro el dibujo rotará constantemente debido al error de bias producido.

---

[17] El programa *Atomic Mixer* se puede descargar de la siguiente página de *Sparkfun*: http://www.sparkfun.com/commerce/product_info.php?products_id=9184

El programa modificado lo podemos ver en la Figura 32 y está detallado en el Anexo A.

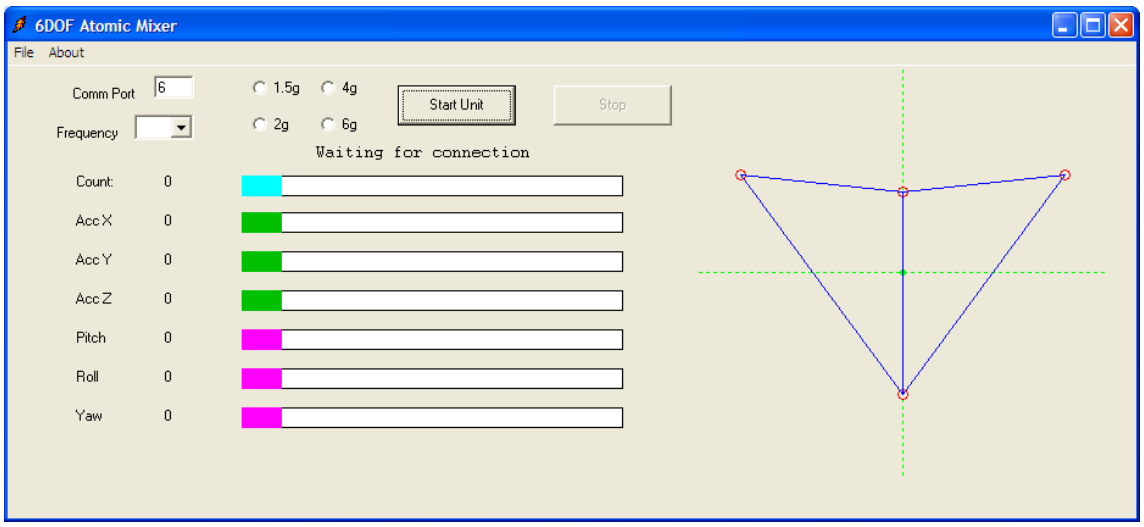

Figura 32. Programa *Atomic Mixer* Adaptado.

4. Conclusiones y líneas futuras

Se han cumplido los objetivos planteados en el trabajo como son la implementación de un programa para la unidad de medida inercial que sincroniza el envío de datos de aceleración y velocidad angular con la FPGA y aprovecha al máximo sus recursos, maximizando la frecuencia de trabajo y facilitando al usuario la configuración de sensibilidad y frecuencia de la IMU. Esto se ha comprobado mediante la aplicación implementada en Visual Basic que nos muestra el movimiento efectuado por la IMU en tiempo real.

Los resultados obtenidos mediante el filtro de Kalman del ángulo han sido los esperados, eliminando el bias producido por los sensores de giro y obteniendo así el ángulo correcto a largos plazos de tiempo. Cabe decir que la corrección del ángulo con el filtro de Kalman solo se ha podido implementar para los ángulos pitch y roll, dejando al ángulo yaw con bias.

Como trabajo futuro podemos plantear que, debido a que el conversor analógico-digital con el que cuenta el microprocesador lo estamos utilizando en serie mediante un multiplexor en lugar de utilizarlo en paralelo, podríamos ganar el tiempo que pierde el conversor al empezar a convertir una nueva muestra si convierte las muestras seguidamente.

5. Referencias Bibliográficas

[1] P. Huerta, "Tutorial EDK 8.2", Universidad Rey Juan Carlos.

[2] A. Yosef, "An Inertial Measurement Unit for User Interfaces", Univertity of British Columbia (1998).

[3] G. Welch and G. Bishop, "An Introduction to the Kalman Filter", Univertity of North Carolina at Chapel Hill.

[4] A. Ramírez y J. Fernández, "Integración GPS/INS: Conceptos y Experimentos", UPC.

Páginas web:

[5] http://tom.pycke.be/mav/71/kalman-filtering-of-imu-data

[6] http://www.todopic.com.ar/foros/index.php?topic=12748.210

[7] http://media.freescale.com/phoenix.zhtml?c=196520&p=irolnewsArticle_Print&ID= 1324107&highlight

[8] http://www.gte.us.es/ASIGN/SEA/MEMS_PRACT.pdf

[9] http://www.gyroscopes.org

[10] http://www.fpga4fun.com

[11] http://usuarios.multimania.es/israelsu/unimayab/Arquitectura%20de%20Computadoras/Arq%20computadoras_04.pdf

Anexo A

ATMega168 IMU Code

```
/*
      6DOF Atomic

      20/6/10, Alfred Raul Giménez Bonastre

      Este programa específico para la Atomic IMU funciona a través de una
      FPGA. La configuración se hace mediante los switch de la FPGA
      (Sensibilidad, Frecuencia) y con el pulsador de la FPGA enviamos el valor
      de los switch para que la unidad se configure.
      Cuando envía datos, la trama tiene la siguiente forma:

      Aaxayazwxwywz , donde a es aceleración y w velocidad angular en ASCII

*/

//Librerías y Definiciones
//=========================================================
#include <avr/io.h>
#include "rprintf.h"
#include <math.h>
#include <avr/interrupt.h>

#define FOSC 10000000              //Velocidad de reloj
#define BAUD 115200                //Baudios
#define MYUBRR FOSC/8/BAUD-1
#define STAT 5

#define x_active          5
#define y_active          4
#define z_active          3
#define pitch_active      2
#define roll_active       1
#define yaw_active        0

#define FREQ_LOW          0
#define FREQ_HIGH         1
#define SENSE_AR_MODE2
#define ACT_CHAN          3

#define GS1 0
#define GS2       1

//Definición de Funciones
//=========================================================
void EEPROM_write(unsigned int uiAddress, unsigned char ucData);
unsigned char EEPROM_read(unsigned int uiAddress);
void ioinit(void);
void USART_Init( unsigned int ubrr);
void Configurar_Imu(char temp1);
void put_char(char byte);
int get_adc(void);
char get_char(void);

void delay_ms(uint16_t x);
void delay_us(uint8_t x);

//Variables Globales
//=========================================================
int x_accel;
int y_accel;
int z_accel;
int pitch;
int roll;
int yaw;

char active_channels = 0b00111111;
float freq = 0;
```

```
int main (void)
{
        char temp, frec, temp2;
        short b;


  ioinit();                       //Inicializa los pines de entrada/salida
        USART_Init(10);           //Inicializa la UART a 115200 baudios
        rprintf_devopen(put_char);

        for (b = 0; b < 5; b++)   //Parpadea el led de estado 5 veces al
                                  // encender
        {
                PORTB &= (~(1<<STAT));          //Se enciende el led
                delay_ms(50);
                PORTB |= (1<<STAT);             //Se apaga el led
                delay_ms(50);
        }
        cli();    //Desabilita interrupciones

        //Frecuencia a 100 Hz
        EEPROM_write((unsigned int) FREQ_LOW, 50);
        EEPROM_write((unsigned int) FREQ_HIGH, 0);
        //Sensibilidad por defecto a 1.5g
        EEPROM_write((unsigned int) SENSE_AR_MODE, 0);
        //Activa todos los canales (sensores)
        EEPROM_write((unsigned int) ACT_CHAN, 0x3F);

        sei();              //Habilita interrupciones

        freq = 100;         //Frecuencia inicial a 100 Hz
        PORTB &= (~((1<<GS1) | (1<<GS2)));          //Sensibilidad a 1.5g

//=====================Bucle Principal=========================
        while(1)
        {
                if (UCSR0A & (1<<RXC0))             //Si la IMU recibe algo...
                {
                    temp = UDR0;
                    temp2 = temp & 0x10;
                     if (temp2== 16) get_adc();     //Si lo que recibe es el
                                                    // pulsador de la FPGA
                }
        }
        while(1);
}

//Inicializamos los pines de entrada/salida y timer
//=========================================================
void ioinit (void)
{
        PORTB |= (1<<STAT);
        DDRB |= ((1<<STAT) | (1<<GS1) | (1<<GS2));

        TCCR1B = (1<<CS10) | (1<<CS11); //Preescaler del Timer a 64
        TCCR2B = (1<<CS21);
}

//Inicializa la UART
//=========================================================
void USART_Init(unsigned int ubrr)
{
        //Inicializa la velocidad de transmisión
        UBRR0H = (unsigned char)(ubrr>>8);
        UBRR0L = (unsigned char)ubrr;
```

```
        // Habilita receptor y transmisor
        UCSR0A = (1<<U2X0);
        UCSR0B = ((1<<RXEN0)|(1<<TXEN0));

        // Inicializa el formato de la trama: 8datos, 2bits de parada
        UCSR0C = ((1<<UCSZ00)|(1<<UCSZ01));
}

//Delays generales en ms
//=========================================================
void delay_ms(uint16_t x)
{
  for (; x > 0 ; x--)
  {
    delay_us(250);          //Llama a la función de delays de us
    delay_us(250);
    delay_us(250);
    delay_us(250);
  }
}

//Delays generales en us aproximadamente
//=========================================================
void delay_us(uint8_t x)
{
        char temp;

        if (x == 0) temp = 1;
        else temp = x;

        TIFR2 |= 0x01;    //Limpia flags de interrupciones en el Timer2
        TCNT2 = 256 - temp;
        while(!(TIFR2 & 0x01));
}
```

```
//Función para enviar un caracter
//=========================================================
void put_char(char byte)
{
        /* Espera a que el buffer de transmisión esté vacío */
        while (!( UCSR0A & (1<<UDRE0)));
        /* Almacena el dato en el buffer y lo envía */
        UDR0 = byte;
}

//Función para recibir un caracter
//=========================================================
char get_char(void)
{
        /* Espera a que el buffer de recepción esté lleno */
        while(!(UCSR0A & (1<<RXC0)));
        return UDR0;
}

//Función que toma los datos de los sensores
//=========================================================
int get_adc(void)
{
        int h2 = 0, l2 = 0;
        int time = 0, tmr_cnt;
        char q, a;

        x_accel = 0;
        y_accel = 0;
        z_accel = 0;
        pitch = 0;
        roll = 0;
        yaw = 0;
```

```
//Contador = (1/Target Frequency)/(Preescaler/Clock Frequency) - 1
tmr_cnt = (int)((1/freq)/.0000064) - 1;

while(1)
{
        TIFR1 |= 0x01;
        TCNT1 = 65536 - tmr_cnt;        //Inicializa el contador para
                                        //cada iteración

        x_accel = 0, y_accel = 0, z_accel = 0;
        pitch = 0, roll = 0, yaw = 0;

        put_char('A');                  //Envía el inicio de trama 'A'

//Conversión aceleración en x============================
        //Selecciona el canal del multiplexor
        ADMUX = 0x40;
        //Activa y configura el conversor A/D
        ADCSRA = (1 << ADEN)|(1 << ADSC)|(1<<ADPS2)|(1<<ADPS1);
        while(ADCSRA & (1 << ADSC));
        l2 = (ADCL & 0xFF);
        h2 = (ADCH & 0x03);
        x_accel = ((h2<<8) | l2);

        rprintf("%d",x_accel);          //Enviamos el dato de ax
        h2 = 0; l2 = 0;

//Conversión aceleración en y============================
        ADMUX = 0x41;
        ADCSRA = (1 << ADEN)|(1 << ADSC)|(1<<ADPS2)|(1<<ADPS1);
        while(ADCSRA & (1 << ADSC));
        l2 = (ADCL & 0xFF);
        h2 = (ADCH & 0x03);
        y_accel = ((h2<<8) | l2);
        rprintf("%d",y_accel);
        h2 = 0; l2 = 0;

//Conversión aceleración en z============================
        ADMUX = 0x42;
        ADCSRA = (1 << ADEN)|(1 << ADSC)|(1<<ADPS2)|(1<<ADPS1);
        while(ADCSRA & (1 << ADSC));
        l2 = (ADCL & 0xFF);
        h2 = (ADCH & 0x03);
        z_accel = ((h2<<8) | l2);

        rprintf("%d",z_accel);
        h2 = 0; l2 = 0;


//Conversion de pitch===================================
        ADMUX = 0x43;
        ADCSRA = (1 << ADEN)|(1 << ADSC)|(1<<ADPS2)|(1<<ADPS1);
        while(ADCSRA & (1 << ADSC));
        l2 = (ADCL & 0xFF);
        h2 = (ADCH & 0x03);
        pitch = ((h2<<8) | l2);

        rprintf("%d",pitch);
        h2 = 0; l2 = 0;

//Conversion de roll====================================
        ADMUX = 0x44;
        ADCSRA = (1 << ADEN)|(1 << ADSC)|(1<<ADPS2)|(1<<ADPS1);
        while(ADCSRA & (1 << ADSC));
        l2 = (ADCL & 0xFF);
        h2 = (ADCH & 0x03);
        roll = ((h2<<8) | l2);
```

```
		rprintf("%d",roll);
		h2 = 0; l2 = 0;

//Conversion de yaw===================================
		ADMUX = 0x45;
		ADCSRA = (1 << ADEN)|(1 << ADSC)|(1<<ADPS2)|(1<<ADPS1);
		while(ADCSRA & (1 << ADSC));
		l2 = (ADCL & 0xFF);
		h2 = (ADCH & 0x03);
		yaw = ((h2<<8) | l2);

		rprintf("%d",yaw);
		h2 = 0; l2 = 0;

//Si el contador termina antes de llegar a este punto significa que la
//frecuencia de trabajo es demasiado alta
		if (TIFR1 & 0x01)
		{
			rprintf("\r\n\nFrecuencia demasiado alta!!\r\n\n",0);
			freq = 100;
			break;
		}

		while(!(TIFR1 & 0x01));	//Espera a que termine el timer

		if (UCSR0A & (1<<RXC0))	//Si la unidad recibe algo...
		{
			q = UDR0;

		//Máscara para saber si hemos pulsado el pulsador de la FPGA
			a = q & 0x10;

		//Entrar en el menú de configuración
			if (a == 16)
				{
			//Si se ha pulsado el pulsador se entra en la configuración
					Configurar_Imu(q);

			//Actualizamos el valor del timer con la nueva frecuencia
					tmr_cnt = (int)((1/freq)/.0000064);
				}
			}
		}
		return;
}

//Función de configuración de la IMU
//=========================================================
void Configurar_Imu(char temp1)
{
		char sens, frec;

		//Máscara para quedarnos con los switchs de la FPGA de sensibilidad
		sens = temp1 & 0xC;
		//Máscara para quedarnos con los switchs de la FPGA de frecuencia
		frec = temp1 & 0x3;

			//-------------------------------------SENSIBILIDAD----------------
			if (sens == 0)				//Sensibilidad 1.5g
			{
				PORTB &= (~((1<<GS1) | (1<<GS2)));//GS1, GS2 Low
			}
			else if (sens == 4)		//Sensibilidad a 2g
			{
				PORTB |= (1<<GS1);	//GS1 High
				PORTB &= (~(1<<GS2));	//GS2 Low
			}
			else if (sens == 8)		//Sensibilidad a 4g
```

```
		{
			PORTB &= (~(1<<GS1));	//GS1 Low
			PORTB |= (1<<GS2);	//GS2 High
		}
		else if (sens == 12)		//Sensibilidad a 6g
		{
			PORTB |= ((1<<GS1) | (1<<GS2));	//GS1, GS2 High
		}

		//----------------------------------------FRECUENCIA---------------
		if (frec == 0)		//Frecuencia a 100Hz
		{
			freq = 100;
		}
		else if (frec == 1) //Frecuencia a 200Hz
		{
			freq = 200;
		}
		else if (frec == 2) //Frecuencia a 250Hz
		{
			freq = 250;
		}
		else if (frec == 3) //Frecuencia a 300Hz
		{
			freq = 300;
		}
}

//Función que escribe en la EEPROM
//=============================================================
void EEPROM_write(unsigned int uiAddress, unsigned char ucData)
{
/* Espera a que se complete la escritura anterior */
while(EECR & (1<<EEPE));
/* Inicializa las direcciones y datos */
EEAR = uiAddress;
EEDR = ucData;
EECR |= (1<<EEMPE);
/* Indica que se ha terminado la escritura */
EECR |= (1<<EEPE);
}

//Función que lee de la EEPROM
//=============================================================
unsigned char EEPROM_read(unsigned int uiAddress)
{
	/* Espera a que se complete la lectura anterior */
	while(EECR & (1<<EEPE));
	/* Inicializa la dirección */
	EEAR = uiAddress;
	EECR |= (1<<EERE);
	return EEDR;
}
```

Anexo B

Código en Visual Basic para la visión en tiempo real del funcionamiento de la IMU. Aplicación del filtro de Kalman

```
'Option Explicit

'6DOF Alfred Giménez

'Variables Globales
Dim Stop_Waiting        As Boolean
Dim CommPort            As Integer
Dim freq                As Integer
Dim low                 As Boolean
Dim med                 As Boolean
Dim high                As Boolean
Dim really_high         As Boolean

Private Const PI = 3.14159265358979
Private Const Vref = 3.3
Private Const g = 9.8
Private Const Sens_acc = 0.8
Private Const Sens_gyro = 0.0033

'Constantes Filtro de Kalman
Private Const R_angle = 0.3
Private Const Q_angle = 0.001
Private Const Q_gyro = 0.003

'Variables utilizadas para mover el dibujo al mover la IMU
Private Enum TipoMovimientoConstants
   tmvLeft
   tmvRight
   tmvUp
   tmvDown
   tmvAgainstClock
   tmvClock
End Enum
```

```
Private Type Punto
   X  As Single
   Y  As Single
   Z  As Single
End Type

Private Type Linea
   IDPto1        As Integer
   IDPto2        As Integer
   Color         As OLE_COLOR
End Type

Private Type Figura
   Centro        As Punto
   Puntos()      As Punto
   Lineas()      As Linea
End Type

Private Figuras           As Figura
Private angulo_pitch      As Double
Private angulo_roll       As Double
Private angulo_yaw        As Double
Private angulo_ax         As Double
Private angulo_ay         As Double
Private angulo_az         As Double
```

```
'Seleccionamos el puerto Comm
Private Sub cboCommPort_Click()

   CommPort = cboCommPort.ListIndex + 1

End Sub
```

```
'Seleccionamos la frecuencia
```

```
Private Sub cboCombo1_Click()

   freq = cboCombo1.List(cboCombo1.ListIndex)

End Sub
```

```
'Si pulsamos el botón de parar
Private Sub cmdBreak_Click()

   Stop_Waiting = True

End Sub
```

```
'Cuando pulsamos el botón de 'Start Unit' para que empiece a funcionar
Private Sub cmdOpenLink_Click()

   Dim i  As Long
   Dim X As Byte
   Dim data_array(20) As Byte

   Dim count      As Long
   Dim magx       As Long
   Dim magy       As Long
   Dim magz       As Long

   Dim accx       As Long
   Dim accy       As Long
   Dim accz       As Long
   Dim ax         As Double
   Dim ay         As Double
   Dim az         As Double

   Dim pitch      As Long
   Dim roll       As Long
   Dim yaw        As Long

'Para calibrar y encontrar el bias
Dim bias_pitch         As Long
Dim bias_roll          As Long
Dim bias_yaw           As Long
Dim bias_ax            As Long
Dim bias_ay            As Long
Dim bias_az            As Long
Dim dt                 As Double


'Variables donde guardaremos la velocidad angular
Dim ang_pitch          As Double
Dim ang_roll           As Double
Dim ang_yaw            As Double
Dim ang_pitch_acc      As Double
Dim ang_roll_acc       As Double
Dim ang_yaw_acc        As Double

'Variables del filtro de Kalman
Dim P_pitch(1 To 2, 1 To 2)      As Double
Dim P_roll(1 To 2, 1 To 2)       As Double
Dim Pdot(7)                      As Double
Dim w_bias_pitch                 As Double
Dim w_bias_roll                  As Double
Dim angulo_error_pitch           As Double
Dim angulo_error_roll            As Double
Dim K_pitch(1)                   As Double
Dim K_roll(1)                    As Double

'Inicializamos la matriz de covarianza P
P_pitch(1, 1) = 1
P_pitch(1, 2) = 0
```

```
P_pitch(2, 1) = 0
P_pitch(2, 2) = 1
P_roll(1, 1) = 1
P_roll(1, 2) = 0
P_roll(2, 1) = 0
P_roll(2, 2) = 1

'Inicializamos los bias
bias_pitch = 520
bias_roll = 498
bias_yaw = 497
bias_ax = 509
bias_ay = 455
bias_az = 537

'Si el puerto está abierto lo cerramos
If MSComm1.PortOpen = True Then MSComm1.PortOpen = False
cmdBreak.Enabled = True
cmdOpenLink.Enabled = False

lblStatus.Caption = "Conectando a la unidad remota"
MSComm1.CommPort = CommPort
MSComm1.InputLen = 1
'If Flow_control Then MSComm1.Handshaking = 0
MSComm1.Handshaking = 1
MSComm1.InputMode = comInputModeText
MSComm1.Settings = "115200,n,8,1"
MSComm1.PortOpen = True

'Seleccionamos la sensibilidad de los acelerómetros en la unidad
If really_high = True Then MSComm1.Output = Chr(40)
If high = True Then MSComm1.Output = Chr(39)
If med = True Then MSComm1.Output = Chr(38)
If low = True Then MSComm1.Output = Chr(37)

  'Seleccionamos la frecuencia en la unidad
  If freq = 50 Then MSComm1.Output = Chr(41)
  If freq = 100 Then MSComm1.Output = Chr(42)
  If freq = 150 Then MSComm1.Output = Chr(43)
  If freq = 200 Then MSComm1.Output = Chr(44)
  If freq = 250 Then MSComm1.Output = Chr(45)
  MSComm1.Output = Chr(35)

  dt = 1 / freq 'Tiempo de muestreo

'---------------------------------Bucle principal----------------------------
SAMPLE_LOOP:
  DoEvents

  'Escaneamos el primer byte de la trama 'A'
  Do While MSComm1.Input <> "A"
    'Miramos si el usuario quiere parar
    If Stop_Waiting = True Then GoTo GET_OUT
    DoEvents
  Loop

  For X = 0 To 13

    Do While MSComm1.InBufferCount = 0
      'Miramos si el usuario quiere parar
      If Stop_Waiting = True Then GoTo GET_OUT
      DoEvents
    Loop

    'Guardamos la lectura en el vector data_array()
    data_array(X) = AscB(MSComm1.Input)
    DoEvents
  Next X
```

```
'Verificamos que el último byte es correcto 'Z'
Do While MSComm1.InBufferCount = 0
   If Stop_Waiting = True Then GoTo GET_OUT
   DoEvents
Loop

'Significa que tenemos una muestra correcta
If MSComm1.Input = "Z" Then
lblStatus.Caption = "Recibiendo Datos!" 'Cambiamos el estado

   'Contador
   count = data_array(0)
   count = count * 256 + data_array(1)
   lblReading(0).Caption = CStr(count)
   bluebar(0).Width = Int((count * whitebar(0).Width) / 32767)
   DoEvents

   'Aceleraciones
   accx = data_array(2)
   accx = accx * 256 + data_array(3)
   lblReading(4).Caption = CStr(accx)
   bluebar(4).Width = Int((accx * whitebar(4).Width) / 1023)
   ax = (accx - bias_ax) * (Vref / 1024) * (g / Sens_acc)
   DoEvents

   accy = data_array(4)
   accy = accy * 256 + data_array(5)
   lblReading(5).Caption = CStr(accy)
   bluebar(5).Width = Int((accy * whitebar(5).Width) / 1023)
   ay = (accy - bias_ay) * (Vref / 1024) * (g / Sens_acc)
   DoEvents

   accz = data_array(6)
   accz = accz * 256 + data_array(7)
   lblReading(6).Caption = CStr(accz)
   bluebar(6).Width = Int((accz * whitebar(6).Width) / 1023)
   az = (accz - bias_az) * (Vref / 1024) * (g / Sens_acc)
   DoEvents

   'Pitch
   pitch = data_array(8)
   pitch = pitch * 256 + data_array(9)
   lblReading(7).Caption = CStr(pitch)
   bluebar(7).Width = Int((pitch * whitebar(7).Width) / 1023)
   ang_pitch = ((pitch - bias_pitch) / 1024) * (Vref / Sens_gyro) * (PI / 180)
   DoEvents

   'Roll
   roll = data_array(10)
   roll = roll * 256 + data_array(11)
   lblReading(8).Caption = CStr(roll)
   bluebar(8).Width = Int((roll * whitebar(8).Width) / 1023)
   ang_roll = ((roll - bias_roll) / 1024) * (Vref / Sens_gyro) * (PI / 180)
   DoEvents

   'Yaw
   yaw = data_array(12)
   yaw = yaw * 256 + data_array(13)
   lblReading(9).Caption = CStr(yaw)
   bluebar(9).Width = Int((yaw * whitebar(9).Width) / 1023)
   ang_yaw = ((yaw - bias_yaw) / 1024) * (Vref / Sens_gyro) * (PI / 180)
   angulo_yaw = ang_yaw * dt
   DoEvents

End If

'--------------------------------KALMAN-----------------------------------
```

```
'Calculo del ángulo con los gyros y considerando el bias obtenido con Kalman
angulo_pitch = dt * (ang_pitch - w_bias_pitch)
angulo_roll = dt * (-ang_roll - w_bias_roll)

Pdot(0) = Q_angle - P_pitch(1, 2) - P_pitch(2, 1)
Pdot(1) = -P_pitch(2, 2)
Pdot(2) = -P_pitch(2, 2)
Pdot(3) = Q_gyro
Pdot(4) = Q_angle - P_roll(1, 2) - P_roll(2, 1)
Pdot(5) = -P_roll(2, 2)
Pdot(6) = -P_roll(2, 2)
Pdot(7) = Q_gyro

'Actualizamos matriz de covarianza
P_pitch(1, 1) = P_pitch(1, 1) + Pdot(0) * dt
P_pitch(1, 2) = P_pitch(1, 2) + Pdot(1) * dt
P_pitch(2, 1) = P_pitch(2, 1) + Pdot(2) * dt
P_pitch(2, 2) = P_pitch(2, 2) + Pdot(3) * dt
P_roll(1, 1) = P_roll(1, 1) + Pdot(4) * dt
P_roll(1, 2) = P_roll(1, 2) + Pdot(5) * dt
P_roll(2, 1) = P_roll(2, 1) + Pdot(6) * dt
P_roll(2, 2) = P_roll(2, 2) + Pdot(7) * dt

'Calculo del ángulo con los acelerómetros
angulo_pitch_acc = -(Atn(ay / Sqr((ax ^ 2) + (az ^ 2))))  'Pitch en radianes
angulo_roll_acc = -(Atn(ax / Sqr((ay ^ 2) + (az ^ 2))))    'Roll en radianes

'Ángulo error
angulo_error_pitch = angulo_pitch_acc - angulo_pitch
angulo_error_roll = angulo_roll_acc - angulo_roll

'Ganancia de Kalman
K_pitch(0) = P_pitch(1, 1) / (R_angle + P_pitch(1, 1))
K_pitch(1) = P_pitch(2, 1) / (R_angle + P_pitch(1, 1))
K_roll(0) = P_roll(1, 1) / (R_angle + P_roll(1, 1))
K_roll(1) = P_roll(2, 1) / (R_angle + P_roll(1, 1))

P_pitch(1, 1) = P_pitch(1, 1) - (K_pitch(0) * P_pitch(1, 1))
P_pitch(1, 2) = P_pitch(1, 2) - (K_pitch(0) * P_pitch(1, 2))
P_pitch(2, 1) = P_pitch(2, 1) - (K_pitch(1) * P_pitch(1, 1))
P_pitch(2, 2) = P_pitch(2, 2) - (K_pitch(1) * P_pitch(1, 2))
P_roll(1, 1) = P_roll(1, 1) - (K_roll(0) * P_roll(1, 1))
P_roll(1, 2) = P_roll(1, 2) - (K_roll(0) * P_roll(1, 2))
P_roll(2, 1) = P_roll(2, 1) - (K_roll(1) * P_roll(1, 1))
P_roll(2, 2) = P_roll(2, 2) - (K_roll(1) * P_roll(1, 2))

'Ángulos finales
angulo_pitch = angulo_pitch + K_pitch(0) * angulo_error_pitch
angulo_roll = angulo_roll + K_roll(0) * angulo_error_roll

'Bias final
w_bias_pitch = w_bias_pitch + K_pitch(1) * angulo_error_pitch
w_bias_roll = w_bias_roll + K_roll(1) * angulo_error_roll

Call Avion 'Llamamos a la función que mueve el avión

GoTo SAMPLE_LOOP

GET_OUT:
MSComm1.Output = Chr(32)
cmdBreak.Enabled = False
cmdOpenLink.Enabled = True
Stop_Waiting = False

'Mensaje diciendo que hemos parado la unidad
MsgBox "Unit Stopped", vbOKOnly
'Cambio de estado
lblStatus.Caption = "Unidad en reposo..."
```

```
    Exit Sub

'Si hay algún error con la conexión
ErrHandler:
    If Err.Number = 8020 Or Err.Number = 8015 Then
        MsgBox "The PIC seems to be using the UART at a different baud rate. Please
power down the PIC or hold it in reset before initiating download."
    ElseIf Err.Number = 8005 Then
        MsgBox "COM" & CommPort & " already open."
    Else
        MsgBox "Error, " & Err.Description, vbOKOnly
        MsgBox Err.Number
    End If

ErrExit:

    'Si el puerto sigue abierto lo cerramos
    If MSComm1.PortOpen = True Then
        MSComm1.PortOpen = False
    End If

    cmdBreak.Enabled = False
    'Cambio de estado a Error
    lblStatus.Caption = "¡¡Error!!"

End Sub
```

```
'Si cerramos el archivo
Private Sub File_Close_Click()
    Unload Me
End Sub
```

```
'Al cargar el formulario (abrir archivo)
Private Sub Form_Load()

'On Error GoTo EH

    For i = 0 To 5
        'bluebar(i).Width = whitebar(i).Width / 2
    Next i

    CommPort = 6
    txtComPort.Text = CommPort

    Exit Sub

EH:
    MsgBox Err.Number & " = " & Err.Description

End Sub
```

```
'Selecciona la sensibilidad del acelerómetro a 1.5g
Private Sub Option1_Click()

    low = True
    med = False
    high = False
    really_high = False

End Sub
```

```
'Selecciona la sensibilidad del acelerómetro a 2g
Private Sub Option2_Click()
    low = False
    med = True
    high = False
    really_high = False
```

```
End Sub
```

```
'Selecciona la sensibilidad del acelerómetro a 4g
Private Sub Option3_Click()
   low = False
   med = False
   high = True
   really_high = False
End Sub
```

```
'Selecciona la sensibilidad del acelerómetro a 6g
Private Sub Option4_Click()
   low = False
   med = False
   high = False
   really_high = True
End Sub
```

```
'Comprueba si el puerto comm que hemos puesto es correcto
Private Sub txtComPort_Change()

'On Error GoTo Error_Handler

   If CInt(txtComPort.Text) > 99 Then txtComPort.Text = "99"

   CommPort = CInt(txtComPort.Text)

   Exit Sub

Error_Handler:
   MsgBox "El número del puerto Comm debe ser entre 1 y 99"
   txtComPort.Text = "1"

End Sub
```

```
'----------------------------------------------FIGURA-----------------------------------------
'Función que calcula la posición de los puntos
Private Function Pow(Number, Exp As Integer) As Double
   Dim auxI As Integer

   Pow = 1
   For auxI = 1 To Exp
      Pow = Pow * Number
   Next auxI
End Function
```

```
'Función que calcula el ángulo de cada punto
Private Function Angle(X As Single, Y As Single)
   Angle = Atn(Y / X) - IIf(X < 0, PI, 0)
End Function
```

```
'Función que nos crea la figura, con sus respectivos puntos y líneas
Private Sub CreaAvion(Fig As Figura)
   ReDim Fig.Puntos(1 To 5)

   'Situamos la figura a la derecha de la pantalla
   Fig.Centro.X = (Me.Width / 4) * 3.15
   Fig.Centro.Y = (Me.Height / 2) * 0.8
   Fig.Centro.Z = 0

   'Fijamos los puntos vértice de la figura
   Fig.Puntos(1).X = -2500
   Fig.Puntos(1).Y = 1500
   Fig.Puntos(1).Z = 0

   Fig.Puntos(2).X = 2500
   Fig.Puntos(2).Y = -1000
   Fig.Puntos(2).Z = 0
```

```
Fig.Puntos(3).X = -2700
Fig.Puntos(3).Y = -1200
Fig.Puntos(3).Z = 2000

Fig.Puntos(4).X = -2700
Fig.Puntos(4).Y = -1200
Fig.Puntos(4).Z = -2000

Fig.Puntos(5).X = -2000
Fig.Puntos(5).Y = -1000
Fig.Puntos(5).Z = 0

'Unimos los vértices para formar la figura
ReDim Fig.Lineas(1 To 9)

Fig.Lineas(1).IDPto1 = 1
Fig.Lineas(1).IDPto2 = 2
Fig.Lineas(1).Color = vbBlue

Fig.Lineas(2).IDPto1 = 1
Fig.Lineas(2).IDPto2 = 5
Fig.Lineas(2).Color = vbBlue

Fig.Lineas(3).IDPto1 = 1
Fig.Lineas(3).IDPto2 = 3
Fig.Lineas(3).Color = vbBlue

Fig.Lineas(4).IDPto1 = 1
Fig.Lineas(4).IDPto2 = 4
Fig.Lineas(4).Color = vbBlue

Fig.Lineas(5).IDPto1 = 2
Fig.Lineas(5).IDPto2 = 3
Fig.Lineas(5).Color = vbBlue

Fig.Lineas(6).IDPto1 = 2
Fig.Lineas(6).IDPto2 = 4
Fig.Lineas(6).Color = vbBlue

Fig.Lineas(7).IDPto1 = 2
Fig.Lineas(7).IDPto2 = 5
Fig.Lineas(7).Color = vbBlue

Fig.Lineas(8).IDPto1 = 3
Fig.Lineas(8).IDPto2 = 5
Fig.Lineas(8).Color = vbBlue

Fig.Lineas(9).IDPto1 = 4
Fig.Lineas(9).IDPto2 = 5
Fig.Lineas(9).Color = vbBlue

    'La giramos PI/2 en yaw para que se oriente hacia
    'adelante según nuestra posición
    angulo_yaw = (PI / 2)

    'Llamamos a la función que nos dibuja la figura
    Call Avion

End Sub
```

```
'Función que nos dibuja la figura anteriormente descrita
Private Sub DibujaFigura(Fig As Figura, Optional Limpiar As Boolean = False)
    Dim auxI As Integer

    'Dibujamos los ejes x e y (y el punto central)
    Me.PSet (Fig.Centro.X, Fig.Centro.Y), vbGreen
    For auxI = 1 To 6
```

```
    Me.Circle (Fig.Centro.X, Fig.Centro.Y), auxI * 5, IIf(Limpiar, Me.BackColor,
vbGreen)
  Next auxI
  Me.DrawStyle = vbDot
  Me.Line (Fig.Centro.X - 2500, Fig.Centro.Y)-(Fig.Centro.X + 2500, Fig.Centro.Y),
vbGreen
  Me.Line (Fig.Centro.X, Fig.Centro.Y - 2500)-(Fig.Centro.X, Fig.Centro.Y + 2500),
vbGreen

  'Dibujamos los puntos vértices
  Me.DrawStyle = vbSolid
  For auxI = LBound(Fig.Puntos) To UBound(Fig.Puntos)
    Me.Circle (Fig.Centro.X + Fig.Puntos(auxI).X, Fig.Centro.Y +
Fig.Puntos(auxI).Y), 60, IIf(Limpiar, Me.BackColor, vbRed)
  Next auxI

  'Dibujamos las líneas
  For auxI = LBound(Fig.Lineas) To UBound(Fig.Lineas)
    Me.Line (Fig.Centro.X + Fig.Puntos(Fig.Lineas(auxI).IDPto1).X, Fig.Centro.Y +
Fig.Puntos(Fig.Lineas(auxI).IDPto1).Y)-(Fig.Centro.X +
Fig.Puntos(Fig.Lineas(auxI).IDPto2).X, Fig.Centro.Y +
Fig.Puntos(Fig.Lineas(auxI).IDPto2).Y), IIf(Limpiar, Me.BackColor,
Fig.Lineas(auxI).Color)
  Next auxI
End Sub
```

```
'Función que mueve la figura entera llamando a la función que mueve los puntos
Private Sub MueveFigura(Fig As Figura)
  Dim auxI As Integer

  For auxI = LBound(Fig.Puntos) To UBound(Fig.Puntos)
    Call MuevePunto(Fig.Puntos(auxI))
  Next auxI
End Sub
```

```
'Función que mueve cada punto de la figura dependiendo del ángulo obtenido
Private Sub MuevePunto(Pto As Punto)
  Dim Angulo  As Double
  Dim xRadio  As Double

  'Calcula el radio y ángulo del punto a mover y a éste le suma el nuevo
  'ángulo. Luego mueve el punto en x, y, z, según convenga.
  If angulo_yaw <> 0 Then
    xRadio = Sqr(Pow(Pto.X, 2) + Pow(Pto.Z, 2))
    Angulo = Angle(Pto.X, Pto.Z)
    Angulo = Angulo + angulo_yaw
    Pto.X = xRadio * Cos(Angulo)
    Pto.Y = Pto.Y
    Pto.Z = xRadio * Sin(Angulo)
  End If

  If angulo_roll <> 0 Then
    xRadio = Sqr(Pow(Pto.Y, 2) + Pow(Pto.Z, 2))
    Angulo = Angle(Pto.Y, Pto.Z)
    Angulo = Angulo + angulo_roll
    Pto.X = Pto.X
    Pto.Y = xRadio * Cos(Angulo)
    Pto.Z = xRadio * Sin(Angulo)
  End If

  If angulo_pitch <> 0 Then
    xRadio = Sqr(Pow(Pto.X, 2) + Pow(Pto.Y, 2))
    Angulo = Angle(Pto.X, Pto.Y)
    Angulo = Angulo + angulo_pitch
    Pto.X = xRadio * Cos(Angulo)
    Pto.Y = xRadio * Sin(Angulo)
    Pto.Z = Pto.Z
  End If
```

```
End Sub
```

```
'Al abrir el archivo crea la figura y la dibuja
Private Sub Form_Activate()

   'Función principal -> Primero creamos los puntos de la figura y luego los dibujamos
   Call CreaAvion(Figuras)

   Call DibujaFigura(Figuras)

End Sub
```

```
'Función que borra la figura, calcula los puntos nuevos y la dibuja
'llamando a las correspondientes funciones
Private Sub Avion()

      'Borramos la figura anterior para dibujar la nueva
      Call DibujaFigura(Figuras, True)

      'Calculamos los nuevos puntos a mover
      Call MueveFigura(Figuras)

      Call DibujaFigura(Figuras)  'Dibujamos la figura nueva

End Sub
```

Anexo C

Código en Matlab para el estudio de los datos de la IMU. Filtro de Kalman

```
%Programa en Matlab que calcula el ángulo sin filtro y con filtro de Kalman para
%unos datos obtenidos de la IMU y guardados en un fichero.

close all
clear all

fd = fopen('New-1.5g-250Hz-pitch-90º.txt','r');

m = fscanf(fd,'%d',[7,inf]);          %Leer los datos del fichero en filas
fclose(fd);                           %Cerrar el fichero

valor = 4;
w_bias = 0;
g = 9.8;                    %Gravedad
dt = 1/250;                 %Frecuencia de muestreo
vref = 3.3;                 %Voltaje de referencia
w_s = 3.3e-3;               %Sensibilidad giro

%Sensibilidad acelerometros
ac_s = 800e-3;   %1g
%ac_s = 600e-3;  %2g
%ac_s = 300e-3;  %4g
%ac_s = 200e-3;  %6g

%Valores datos filtro Kalman
P = [1,0;0,1];              %Matriz de covarianza
R_angle = 0.3;
Q_angle = 0.001;
Q_gyro = 0.003;

%Velocidad angular
wp1 = m(5,:);
wr = m(6,:);
wy = m(7,:);

%Aceleración
ax = m(2,:);
ay = m(3,:);
az = m(4,:);

%Calculo de la aceleración
ax = (((ax - 509) * (vref / 1024))./ac_s) * g;
ay = (((ay - 455) * (vref / 1024))./ac_s) * g;
az = (((az - 537) * (vref / 1024))./ac_s) * g;

%Calculo de la velocidad angular
wp = ((wp1.*vref/1024)-(vref/2))./w_s;
wr = ((wr.*vref/1024)-(vref/2))./w_s;
wy = ((wy.*vref/1024)-(vref/2))./w_s;

%Vector de datos
dat(1,:) = ax;
dat(2,:) = ay;
dat(3,:) = az;
dat(4,:) = wp;
dat(5,:) = wr;
dat(6,:) = wy;

%Valor inicial del angulo
angulo_wk(1) = dat(valor,1)*dt;
angulo_w(1) = dat(valor,1)*dt;

%Calculamos el valor del angulo y aplicamos el filtro de Kalman
for i=2:length(dat),

  %Angulo sin filtro de Kalman
   angulo_w(i) = angulo_w(i-1) + dat(valor,i)*dt;
   %Angulo con filtro de Kalman
    angulo_wk(i) = angulo_wk(i-1) + dt*(dat(valor,i)-w_bias);
```

```
    Pdot = [Q_angle - P(1,2) - P(2,1), -P(2,2), -P(2,2), Q_gyro];

    %Actualizamos matriz de covarianza
    P(1,1) = P(1,1) + Pdot(1)*dt;
    P(1,2) = P(1,2) + Pdot(2)*dt;
    P(2,1) = P(2,1) + Pdot(3)*dt;
    P(2,2) = P(2,2) + Pdot(4)*dt;

    %Angulo extraído de los acelerómetros
    angulo_a(i) = -(atan(ay(i) / sqrt((ax(i))^2 + (az(i))^2 ))) * (180 / pi);   %Pitch
    angulo_a(i) = -(atan(ax(i) / sqrt((ay(i))^2 + (az(i))^2 ))) * (180 /pi);    %Roll
    angulo_error = angulo_a(i) - angulo_wk(i);

    E = R_angle + P(1,1);
    K0 = P(1,1) / E;
    K1 = P(2,1) / E;

    t_0 = P(1,1);
    t_1 = P(1,2);

    P(1,1) = P(1,1) - (K0 * t_0);
    P(1,2) = P(1,2) - (K0 * t_1);
    P(2,1) = P(2,1) - (K1 * t_0);
    P(2,2) = P(2,2) - (K1 * t_1);

    %Valor final del ángulo y del bias obtenidos a partir de la ganancia de
    %Kalman y del error
    angulo_wk(i) = angulo_wk(i) + K0 * angulo_error;
    w_bias = w_bias + K1 * angulo_error;

end
```

```
%Dibujamos los resultados obtenidos
subplot(1,2,1);
plot(angulo_w);
title('SIN FILTRO DE KALMAN');
xlabel('Muestras');
ylabel('Ángulo');
subplot(1,2,2);
plot(angulo_wk);
title('CON FILTRO DE KALMAN');
xlabel('Muestras');
ylabel('Ángulo');
```

Anexo D

Código para programar la FPGA con EDK

```
/*
 * Xilinx EDK 10.1 EDK_K.15
   Programa para la FPGA Xilinx Virtex-II Pro que lee los datos de la IMU y envía una trama de configuración dependiendo de los switches cuando pulsamos el pulsador central.
*/

// Located in: ppc405_0/include/xparameters.h
#include "xparameters.h"
#include "stdio.h"
#include "xutil.h"
#include "xgpio_l.h"
#include "xuartlite_l.h"

//Define functions
int send_uart(void);
int read_uart(void);


//======================FUNCIÓN PRINCIPAL=========================
int main (void) {

    char pulsar;

    while(1)        //Bucle infinito
    {
        //Si hay datos en el recpetor debemos leer de la IMU
        if(!XUartLite_mIsReceiveEmpty(XPAR_XPS_UARTLITE_0_BASEADDR))
        {
                read_uart();                    //Función que lee los datos de la IMU
        }

        //Leemos el pulsador para saber si queremos configurar la IMU
        pulsar = XGpio_mReadReg(XPAR_PUSHBUTTONS_5BIT_BASEADDR, 0);
        //Máscara para quedarnos con el switch central
        pulsar = pulsar & 0x00000010;

        //Si hemos pulsado el pulsador central
        if(!pulsar)
        {
                send_uart();        //Función que configurará la IMU
        }
    }
    return 0;
}

//-----------------------------FUNCIÓN QUE LEE DATOS DE LA IMU-----------------
int read_uart(void)
{
        char lectura_imu;

        //Leemos datos IMU
        lectura_imu  =  XUartLite_RecvByte(XPAR_XPS_UARTLITE_0_BASEADDR);

        //Sincronizamos receptor
        while(XUartLite_mIsReceiveEmpty(XPAR_XPS_UARTLITE_0_BASEADDR));

        //Enviamos al PC la lectura
        XUartLite_SendByte(XPAR_RS232_UART_1_BASEADDR, lectura_imu);

        //Sincronizamos transmisor
        while(XUartLite_mIsTransmitFull(XPAR_RS232_UART_1_BASEADDR));

}
```

```
//---------------------FUNCIÓN QUE CONFIGURA LA IMU----------------
int send_uart(void)
{
	char interruptores, config;

	//Leemos los interruptores
	interruptores = XGpio_mReadReg(XPAR_DIPSWS_4BIT_BASEADDR, 0);

	//Para cada posible caso...
	switch (interruptores)
	{
	  case 15:
	  config = 0x10;

	  //Enviamos a la UART
	  XUartLite_SendByte(XPAR_XPS_UARTLITE_0_BASEADDR, config);
	  while(XUartLite_mIsTransmitFull(XPAR_XPS_UARTLITE_0_BASEADDR));
	  break;

	  case 14:
	  config = 0x11;

	  XUartLite_SendByte(XPAR_XPS_UARTLITE_0_BASEADDR, config);
	  while(XUartLite_mIsTransmitFull(XPAR_XPS_UARTLITE_0_BASEADDR));
	  break;

	  case 13:
	  config = 0x12;

	  XUartLite_SendByte(XPAR_XPS_UARTLITE_0_BASEADDR, config);
	  while(XUartLite_mIsTransmitFull(XPAR_XPS_UARTLITE_0_BASEADDR));
	  break;

	  case 12:
	  config = 0x13;

	  XUartLite_SendByte(XPAR_XPS_UARTLITE_0_BASEADDR, config);
	  while(XUartLite_mIsTransmitFull(XPAR_XPS_UARTLITE_0_BASEADDR));
	  break;

	  case 11:
	  config = 0x14;

	  XUartLite_SendByte(XPAR_XPS_UARTLITE_0_BASEADDR, config);
	  while(XUartLite_mIsTransmitFull(XPAR_XPS_UARTLITE_0_BASEADDR));
	  break;

	  case 10:
	  config = 0x15;

	  XUartLite_SendByte(XPAR_XPS_UARTLITE_0_BASEADDR, config);
	  while(XUartLite_mIsTransmitFull(XPAR_XPS_UARTLITE_0_BASEADDR));
	  break;

	  case 9:
	  config = 0x16;

	  XUartLite_SendByte(XPAR_XPS_UARTLITE_0_BASEADDR, config);
	  while(XUartLite_mIsTransmitFull(XPAR_XPS_UARTLITE_0_BASEADDR));
	  break;

	  case 8:
	  config = 0x17;

	  XUartLite_SendByte(XPAR_XPS_UARTLITE_0_BASEADDR, config);
	  while(XUartLite_mIsTransmitFull(XPAR_XPS_UARTLITE_0_BASEADDR));
	  break;
```

```
case 7:
config = 0x18;

XUartLite_SendByte(XPAR_XPS_UARTLITE_0_BASEADDR, config);
while(XUartLite_mIsTransmitFull(XPAR_XPS_UARTLITE_0_BASEADDR));
break;

case 6:
config = 0x19;

XUartLite_SendByte(XPAR_XPS_UARTLITE_0_BASEADDR, config);
while(XUartLite_mIsTransmitFull(XPAR_XPS_UARTLITE_0_BASEADDR));
break;

case 5:
config = 0x1A;

XUartLite_SendByte(XPAR_XPS_UARTLITE_0_BASEADDR, config);
while(XUartLite_mIsTransmitFull(XPAR_XPS_UARTLITE_0_BASEADDR));
break;

case 4:
config = 0x1B;

XUartLite_SendByte(XPAR_XPS_UARTLITE_0_BASEADDR, config);
while(XUartLite_mIsTransmitFull(XPAR_XPS_UARTLITE_0_BASEADDR));
break;

case 3:
config = 0x1C;

XUartLite_SendByte(XPAR_XPS_UARTLITE_0_BASEADDR, config);
while(XUartLite_mIsTransmitFull(XPAR_XPS_UARTLITE_0_BASEADDR));
break;

case 2:
config = 0x1D;

XUartLite_SendByte(XPAR_XPS_UARTLITE_0_BASEADDR, config);
while(XUartLite_mIsTransmitFull(XPAR_XPS_UARTLITE_0_BASEADDR));
break;

case 1:
config = 0x1E;

XUartLite_SendByte(XPAR_XPS_UARTLITE_0_BASEADDR, config);
while(XUartLite_mIsTransmitFull(XPAR_XPS_UARTLITE_0_BASEADDR));
break;

default:
config = 0x1F;

XUartLite_SendByte(XPAR_XPS_UARTLITE_0_BASEADDR, config);
while(XUartLite_mIsTransmitFull(XPAR_XPS_UARTLITE_0_BASEADDR));
}
}
```

Resum:

Els sistemes automatitzats que requereixen d'un control d'estabilitat o moviment es poden trobar cada cop en més àmbits. Aplicacions UAV o de posicionament global són les més comunes per aquest tipus de sistemes, degut a que necessiten d'un control de moviment molt precís. Per a dur a terme aquest procés s'utilitzen unitats de mesura inercial, que mitjançant acceleròmetres i giroscopis degudament posicionats, a més a més d'una correcció del possible error que puguin introduir aquests últims, proporcionen una acceleració i una velocitat angular de les quals es pot extreure el camí efectuat per aquestes unitats.

La IMU, combinada amb un GPS i mitjançant un filtre de Kalman, proporcionen una major exactitud , a més d'un punt de partida (proporcionat per el GPS), un recorregut representable en un mapa y, en el cas de perdre la senyal GPS, poder seguir adquirint dades de la IMU. Aquestes dades poden ser recollides i processades per una FPGA, que a la vegada podem sincronitzar amb una PDA per a que l'usuari pugui veure representat el moviment del sistema.

Aquest treball es centra en el funcionament de la IMU i l'adquisició de dades amb la FPGA. També introdueix el filtre de Kalman per a la correcció de l'error dels sensors.

Resumen:

Los sistemas automatizados que requieren de un control de estabilidad o movimiento se encuentran cada vez en más ámbitos. Aplicaciones UAV o de posicionamiento global son las más comunes para este tipo de sistemas, ya que necesitan de un control de movimiento muy preciso. Para ello utilizan unidades de medida inercial, las cuáles, mediante unos acelerómetros y unos giróscopos debidamente posicionados y una corrección del posible error que pueda haber en éstos, proporcionan una aceleración y velocidad angular de las que puedes extraer el recorrido sufrido por dichas unidades.

La IMU, en combinación con un GPS y mediante un filtro de Kalman, proporcionan una exactitud mayor, además de tener un punto de partida (proporcionado por el GPS), un recorrido representable en un mapa y, en caso de perder la señal GPS, seguir adquiriendo datos de la IMU. Estos datos pueden ser recogidos y procesados por una FPGA, la cual se puede sincronizar con una PDA para que el usuario pueda ver representado el movimiento del sistema.

Este trabajo estudia el funcionamiento de la IMU y la adquisición de datos de ésta con la FPGA. También introduce el filtro de Kalman para la corrección del error de los sensores.

Summary:

The automated systems that need a control of stability or motion control are increasingly in more areas. UAV or global positioning applications are the most common for these systems because they need a very precise motion control. They use inertial measurement units, which, with accelerometers and gyroscopes properly positioned and correction of possible errors that may appear in them, provide an acceleration and angular velocity that you can extract the movement suffered by these units.

The IMU, in combination with GPS and through a Kalman filter, provide greater accuracy, in addition to a starting point (provided by GPS), a representable route map and if you lose the GPS signal, the system continues acquiring data of IMU. These data can be processed by an FPGA, which can synchronize with a PDA. In this way, the user can see the motion of the system represented in a map.

This works study the functionality of the IMU and the data acquisition with the FPGA. It also introduces the Kalman filter to correct the error of the sensors.