Memorias del Congreso Nacional de Control Automático
ISSN: 2594-2492

Control en Tiempo Real de un Cuatrirotor Empleando una Computadora en Placa Reducida

Alejandro Gómez-Casasola * **Hugo Rodríguez-Cortés** **

* *Centro de Investigación y de Estudios Avanzados del IPN, México (e-mail: a_gc@live.com.mx)*
** *Centro de Investigación y de Estudios Avanzados del IPN, México (e-mail: hrodriguez@cinvestav.mx)*

Abstract: En este artículo se presenta la puesta en funcionamiento de un cuatrirotor que utiliza una computadora en placa reducida como procesador central. Para verificar el funcionamiento del cuatrirotor se implementa en tiempo real un algoritmo de control no lineal tomado de la literatura. Se abordan consideraciones importantes respecto a la comunicación de la computadora en placa reducida con los sensores y actuadores así como la ejecución en tiempo real del algoritmo de control.

Keywords: Cuatrirotor, microprocesador, sistema en tiempo real, control no lineal, control de orientación, control de posición.

1. INTRODUCCIÓN

El helicóptero impulsado por cuatro hélices, o cuatrirotor, es un vehículo aéreo no tripulado, o UAV por sus siglas en inglés, que en los últimos años ha tenido una gran importancia y aceptación tanto en el sector de investigación como en la industria. Esto gracias a su dinámica que le permite hacer diferentes maniobras de vuelo así como su relativa facilidad de controlarlo.

Por la parte de investigación se han desarrollado diferentes esquemas de control tanto lineales, ver Erginer and Altug (2007), como no lineales, ver Mian and Daobo (2008).

Por otro lado, en la industria se le han dado diferentes aplicaciones para el beneficio humano como son la entrega de paquetes, agricultura de precisión, control de incendios forestales, investigaciones arqueológicas, búsqueda de personas, etc., ver Gupte et al. (2012).

Los primeros cuatrirotores se implementaron con microcontroladores (MCU) o procesadores digital de señales (DSP) encargados de llevar a cabo los cálculos necesarios para el algoritmo de control.

En Guadarrama-Olvera et al. (2014) se presenta la implementación en tiempo real estricto de un algoritmo de control no lineal para resolver los problemas de regulación y seguimiento de trayectorias en un cuatrirotor utilizando un DSP.

Sin embargo, como se mencionó anteriormente, debido a su gran aceptación y desarrollo que han tenido los cuatrirotores, se les han encontrado otras áreas de aplicación más complejas como el aprendizaje máquina y el vuelo completamente autónomo; y los algoritmos de control también han ido mejorando y volviéndose más complejos como es el caso de controladores basados en mediciones a partir de cámaras y algoritmos de visión computacional, ver Garcia et al. (2015), o controladores basados en algoritmos de fusión de sensores, ver Lynen et al. (2013), etc.

Estas nuevas tareas y aplicaciones en los cuatrirotores hacen imprescindible la implementación de un procesador con un mayor nivel de cómputo que sea capaz de llevar a cabo todos los procesos antes mencionados y afines.

Es por estas razones que en el presente artículo se muestra la construcción de un cuatrirotor experimental con un microprocesador (MPU) como procesador central ejecutando un sistema operativo y las consideraciones para lograr una ejecución en tiempo real.

En la sección 2 se menciona el hardware necesario y su interconexión. En la parte 3 se explica un método para lograr una ejecución en tiempo real. En el apartado 4 se detalla una estrategia de control para regulación y seguimiento de trayectorias. En la sección 5 se realiza un vuelo experimental y se reportan los resultados obtenidos, y finalmente, en el apartado 6 se mencionan las conclusiones del artículo.

2. DISEÑO DE HARDWARE

2.1 Procesador

Se escogió como procesador central la tarjeta Raspberry Pi 3, la cual es una tarjeta de desarrollo o computadora en placa reducida (single board computer) que cuenta con un microprocesador embebido y todos los periféricos

Fig. 1. Raspberry Pi 3

necesarios para su uso, como son memoria RAM, memoria no volátil, conexiones de entrada y salida, etc.

Se decidió utilizar esta tarjeta debido a que es muy popular en el ámbito de desarrolladores por lo que se tiene mucha información disponible sobre su uso, además que es de fácil adquisición y maneja una lógica de 3.3V la cual es compatible con el restos de componentes a utilizar en este artículo, a diferencia de la tarjeta ODROID-XU4, por ejemplo, la cual maneja una lógica de 1.8V.

En la Fig. 1 se muestra la tarjeta Raspberry Pi 3 y en la Tabla 1 sus especificaciones técnicas.

Tabla 1. Especificaciones técnicas de la Raspberry Pi 3

Microprocesador	BCM2837 64bit Quad-Core 1.2GHz
GPU	VideoCore IV 400MHz
Memoria	1GB RAM
Almacenamiento	Hasta 32GB
Comunicación inalámbrica	WiFi y Bluetooth 4.1
Puertos de expansión	UART, I2C, SPI, GPIO
Sistema Operativo	Raspbian, Ubuntu

2.2 Actuadores e interfaz de potencia

Como actuadores para impulsar el cuatrirotor se utilizaron los motores T-Motor MN2213-12, los cuales incluyen su juego de hélices y controladores electrónicos de velocidad (ESC). Estos últimos por lo general son controlados por una señal lógica de entre 1 y 2 milisegundos proveniente del procesador central para obtener el empuje deseado en cada motor que normalmente se implementa con una señal PWM.

2.3 Módulo PWM

Uno de los detalles más importantes a tener en cuenta con los MPU es que tienen una arquitectura diferente a los MCU o DSP, como es el hecho de que los MPU no cuentan con módulos PWM internos, que es el medio que normalmente se ocupa para controlar los motores por medio de los ESC. Debido a esto es necesario implementar un módulo PWM externo.

Primeramente, se hizo la prueba con el circuito integrado PCA9685, que es uno de los módulos PWM externos más comunes, el cual tiene una resolución de 12 bits, 16 canales, frecuencia PWM de hasta 1.6 KHz y se comunica con el procesador maestro por medio de I^2C.

Haciendo pruebas experimentales con este módulo, se llegó a la conclusión de que la resolución de 12 bits no es suficiente para tener un control adecuando de los motores, ya que el cuatrirotor nunca se logró estabilizar en vuelo suspendido (hover) con controladores que ya se habían implementado en un DSP con PWM de resolución de 16 bits, ver Vásquez-Beltrán and Rodríguez-Cortés (2015), Guadarrama-Olvera et al. (2014). Debido a esto, se optó por descartar el módulo PCA9685 y buscar una manera de implementar un PWM de 16 bits.

La solución que se dio fue utilizar un microcontrolador que contara con un módulo PWM interno de 16 bits con al menos 4 canales, y se comunicara con el procesador maestro por I^2C, esto es, que el microcontrolador esté programado para hacer exactamente la misma tarea que el integrado PCA9685 pero con un PWM de 16 bits. El microcontrolador que se utilizó fue el STM32F042K6 implementado en la tarjeta de desarrollo NUCLEO-F042K6 el cual cuenta con las especificaciones necesarias mencionadas anteriormente y la tarjeta de desarrollo es de tamaño compacto similar a la tarjeta de pruebas del PCA9685.

2.4 Unidad de medición inercial (IMU)

Para poder realizar el control de orientación, indispensable para cualquier tipo de vuelo manual o automático, es necesario disponer de una serie de sensores que midan el estado actual del sistema.

Para medir la velocidad angular de un cuerpo se utilizan los giróscopos, mientras que para medir la orientación como tal se utiliza un algoritmo de fusión de sensores que combina las mediciones provenientes de los acelerómetros, magnetómetros y giróscopos.

Se utilizó la unidad de medición inercial (IMU) Xsens MTi-3 la cual es un módulo que contiene estos tres sensores, además de un procesador embebido que realiza el algoritmo de fusión, por lo que esta IMU ya entrega directamente las mediciones de aceleración, velocidad angular, campo magnético y orientación en ángulos de Euler, cuaternos o matriz de rotación.

Otra ventaja importante de esta IMU es que tiene la opción de comunicación con el procesador maestro por medio de los protocolos I^2C y SPI. Al ser protocolos estrictamente esclavo-maestro se tienen un total control del flujo de datos ya que la IMU no enviará información a menos que el maestro la solicite.

Anteriormente, las IMU de Xsens y otros fabricantes utilizaban protocolos como RS-232 o UART, en donde la comunicación es asíncrona y en estos casos la IMU envía datos constantemente sin parar, por lo que el maestro tiene que realizar un proceso de sincronización y constantemente estar verificando que esta sincronización no se pierda, como se realiza en Guadarrama-Olvera et al. (2014). Gracias a la implementación de comunicación I^2C o SPI se ahorra este trabajo de sincronización al procesador maestro.

2.5 Posicionamiento

Además del control de orientación, si se desea hacer un vuelo suspendido automático o un seguimiento de trayectoria, se necesita de un control de posición.

Para dicho control es necesario medir los estados actuales de posición del cuatrirotor. Para lograr esto se utiliza un sistema de captura de movimiento (motion capture) basado en cámaras y algoritmos de visión computacional que calculan la posición de un cuerpo rígido a partir de una posición inicial. Esto con la ayuda de marcadores montados sobre el cuerpo los cuales son altamente reflejantes para que las cámaras lo puedan detectar con mayor facilidad y descarten el resto de la imagen que no es tan reflejante.

El sistema de captura de movimiento disponible para este trabajo se conoce comercialmente como OptiTrack, el cual cuenta con un software llamado Motive que realiza todo el trabajo mencionado.

Para enviar la información de la posición del OptiTrack al cuatrirotor se diseñó un programa que manda estos datos por medio de WiFi desde la computadora donde se está ejecutando Motive hasta la Raspberry Pi, recordando que ésta cuenta con comunicación WiFi.

2.6 Alimentación

Los cuatrirotores comúnmente son alimentados por baterías Li-Po de 3 celdas y de una capacidad amper-hora (Ah) dependiendo de la autonimía que se busque y el peso que pueda levantar el cuatrirotor.

En este caso, debido a que se tienen componentes de diferentes requerimentos de voltaje, lo que se hizo fue diseñar una fuente conmutada que se encargue de bajar los 11.1V de la batería a 5V continuos para alimentar la Raspberry Pi, y ésta a su vez genera una salida de 3.3V para alimentar la IMU y el MCU para el PWM.

El hecho de que sea una fuente conmutada asegura que el voltaje de salida será constante a pesar de que el voltaje de entrada esté variando, además que proporciona una mayor eficiencia que una fuente lineal.

La fuente está construida en base al circuito integrado LM2576 el cual soporta un flujo de corriente de hasta 3A, lo cual es más que suficiente para alimentar la Raspberry y los demás componentes lógicos.

De esta forma termina la construcción del cuatrirotor. En la Fig. 2 se muestra el cuatrirotor terminado. Las esferas blancas son los marcadores mencionados en la sección 2.5, y en la Fig. 3 se muestra un diagrama de conexión de todo el sistema.

3. EJECUCIÓN EN TIEMPO REAL

Un requisito muy importante a la hora de implementar un sistema de control es que el procesador que lo esté ejecutando tenga bien definido el tiempo de muestreo de cada iteración, y más importante aún que el procesador ejecute los procesos verdaderamente en ese tiempo.

Fig. 2. Cuatrirotor controlado con Raspberry Pi 3

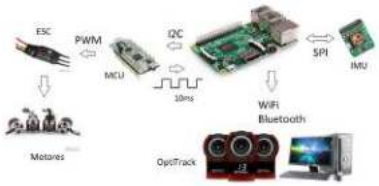

Fig. 3. Diagrama de conexión

Algunos ejemplos de esta necesidad es cuando se requiere realizar una integración o derivación numérica, ya que en estas operaciones se requiere conocer el tiempo de muestreo. O cuando se requiere hacer adquisición de mediciones hechas por el cuatrirotor, es necesario conocer en qué tiempo se realizó dicha medición. Es por estas razones que el procesador de un cuatrirotor debe tener una ejecución en tiempo real.

Para el caso de un MCU o DSP, esta tarea es fácil de realizar ya que existen modelos de programación como Giotto, ver Henzinger et al. (2003), así como sistemas operativos en tiempo real (RTOS) disponibles para estos dispositivos, los cuales se encargan de realizar toda la programación de bajo nivel para que el dispositivo se ejecute respetando los tiempos y tareas que el usuario indique.

Sin embargo, como se mencionó anteriormente, un MPU tiene una arquitectura diferente a éstos. Una de las diferencias más importantes es que los MPU ejecutan un sistema operativo específico para su funcionamiento, como Windows o Linux. Estos sistemas operativos no fueron pensados para lograr una ejecución en tiempo real, y en caso de quitarlo e implementar un RTOS se perdería toda la gestión de controladores (drivers) y subprogramas para el acceso a la mayoría del hardware, como puertos USB, comunicación inalámbrica, etc., por lo que se tiene que buscar otra alternativa para alcanzar dicho fin.

La primera de las opciones que se intentaron fue implementar un parche llamado RT-Preempt Patch el cual hace una modificación al núcleo de Linux para que el sistema otorge la mayor de las prioridades a las interrupciones programadas por el usuario, y deje en segundo plano a todas las tareas requeridas por el mismo sistema operativo.

Esta opción en principio brindó resultados aceptables ya que logró una ejecución en tiempo real conforme lo especificaba el usuario, y la forma de constatarlo fue programando uno de los puertos GPIO de la Raspberry al inicio y final de cada una de las tareas y medir con un osciloscopio el tiempo del pulso generado.

Sin embargo, el problema que presentó esta opción fue que, por razones hasta la fecha desconocidas, al instalar este parche en la Raspberry Pi quedó completamente inhabilitada la comunicación WiFi, la cual como se mencionó anteriormente es necesaria para el posicionamiento del cuatrirotor con el OptiTrack.

Además de esto, se observó un incremento notable en la temperatura del procesador al estar ejecutando los programas bajo este parche, hasta el grado de apagarse la tarjeta por sí misma debido a la alta temperatura en tiempos prolongados.

En la página oficial de la Fundación Linux se muestran los problemas conocidos que genera este parche sobre la Raspberry Pi, entre ellos que los drivers USB quedan deshabilitados, pero no se menciona nada sobre la comunicación WiFi.

La siguiente opción que se intentó fue generar interrupciones internas por medio de rutinas de tiempo usando los timers del MPU. Al ver el comportamiento en el osciloscopio se observó que no es una solución viable ya que los tiempos se perturban fácilmente con cualquier tarea que realice el sistema operativo, incluso sin manipular en lo absoluto a la Raspberry Pi.

Finalmente, lo que se ideó fue generar interrupciones de forma externa con algún dispositivo que estuviera mandando pulsos del tiempo de muestreo exacto directamente a uno de los pines GPIO de entrada y utilizar las funciones de interrupción de la librería wiringPi para detectar dichos pulsos. Estas funciones tienen la opción de ser programadas para detectar flancos de subida, de bajada, o ambos.

El dispositivo que se utilizó para mandar los pulsos fue el mismo MCU utilizado para generar las señales PWM especificadas anteriormente.

Al ver la ejecución de esta opción en el osciloscopio, se observó que los tiempos se mantenían estables y sólo sufría de ligeras perturbaciones en los casos cuando se movía el mouse, el teclado o se abría algún programa en paralelo. Sin embargo, en la ejecución real de algún programa para el cuatrirotor, no se llevarán a cabo ninguna de estas acciones, por lo que los tiempos se mantendrán estables, por lo que esta opción es una solución temporal y la utilizada en este documento para lograr una ejecución en tiempo real, aunque considerando que no es un tiempo real totalmente estricto.

4. CONTROL

Una vez teniendo el hardware completo y funcionando en tiempo real, lo siguiente es implementar un algoritmo de control para poder realizar vuelos experimentales. Como se mencionó anteriormente, es indispensable la implementación de un control de orientación para poder realizar vuelos al menos controlados manualmente, mientras que para realizar vuelos autónomos sobre una trayectoria definida se necesita, además de éste, un control de posición.

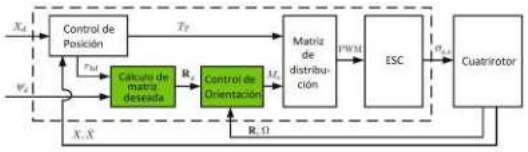

Fig. 4. Diagrama de control

En este apartado se presenta un control de posición y orientación para poder realizar tanto vuelos suspendidos (regulación) como seguimientos de trayectorias. El desarrollo de este controlador así como el análisis de estabilidad se pueden consultar en Tlatelpa-Osorio (2017).

Este controlador está diseñado en el espacio $SO(3)$, por lo que no están presentes los problemas de singularidades como en la representación de ángulos de Euler o ambigüedad en los cuaternos.

En la Fig. 4 se muestra el diagrama de control, las entradas que necesita cada bloque y las salidas que proporcionan.

4.1 Control de posición

Partiendo de las ecuaciones de movimiento de un cuatrirotor, esto es, su dinámica rotacional:

$$J\dot{\Omega} = -\hat{\Omega}J\Omega + M_e \tag{1}$$

la cinemática de la matriz de rotación:

$$\dot{R} = R\hat{\Omega} \tag{2}$$

y la dinámica traslacional en ejes inerciales, modeladas como un cuerpo rígido:

$$m\ddot{X} = mge_3 - T_T Re_3 \tag{3}$$

donde m es la masa del cuatrirotor, g la aceleración de la gravedad, T_T el empuje total de los 4 motores, X es el vector de posiciones, R es la matriz de rotación de ejes cuerpo a ejes inerciales, $e_3 = [0\ 0\ 1]^\top$, J es la matriz de inercia, Ω el vector de velocidades angulares, M_e el vector de momentos de inercia y $\hat{\Omega}$ la matriz antisimétrica dependiente de las velocidades angulares definida como:

$$\hat{\Omega} = \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix}$$

Los espacios de configuración de cada una de las variables son $T_T \in \mathbb{R}$, $\Omega, \dot{\Omega}, M_e, X, \dot{X}, \ddot{X} \in \mathbb{R}^3$ y $R \in SO(3)$.

Teniendo los estados deseados X_d y $\dot{X}_d$ definidos, el control de posición PD queda como:

$$u = m(K_P\tilde{X} + K_D\dot{\tilde{X}} + ge_3 - \ddot{X}_d) \tag{4}$$

donde K_P y K_D son ganancias, $\tilde{X} = X - X_d$ el error de posición y $\dot{\tilde{X}} = \dot{X} - \dot{X}_d$ el error en velocidad.

Para obtener la señal de control T_T se utiliza la tercer columna de la matriz de rotación:

$$T_T = u^\top r_3 \tag{5}$$

A continuación, se calcula la matriz de rotación deseada R_d necesaria para el control de orientación, usando el ángulo de guiñada deseado ψ_d.

$$r_{3d} = \frac{u}{\|u\|} \tag{6}$$

$$\bar{r}_{1d} = \begin{bmatrix} cos(\psi_d) \\ sin(\psi_d) \\ 0 \end{bmatrix} \tag{7}$$

$$r_{2d} = \frac{r_{3d} \times \bar{r}_{1d}}{\|r_{3d} \times \bar{r}_{1d}\|} \tag{8}$$

$$r_{1d} = \frac{r_{2d} \times r_{3d}}{\|r_{2d} \times r_{3d}\|} \tag{9}$$

$$R_d = [r_{1d}\ r_{2d}\ r_{3d}\ ] \tag{10}$$

4.2 Control de orientación

El control de orientación queda definido de la siguiente forma:

$$M_e = \hat{\Omega} J \Omega - K_{P_R} e_R - K_{D_R} \tilde{\Omega} - J[\hat{\Omega} E^\top \Omega_d - E^\top \dot{\Omega}_d] \tag{11}$$

donde K_{P_R} y K_{D_R} son ganancias, e_R el error de orientación y $\tilde{\Omega}$ el error en velocidad angular, los cuales se definen como:

$$e_R = \frac{1}{2}(E - E^\top)^\vee \tag{12}$$

$$\tilde{\Omega} = \Omega - E^\top \Omega_d \tag{13}$$

$$E = R_d^\top R \tag{14}$$

4.3 Sintonización

Finalmente, se necesita encontrar el valor de las ganancias K_P, K_D, K_{P_R} y K_{D_R} para que el controlador funcione correctamente y pueda realizar el vuelo suspendido y seguimiento de trayectorias sin desestabilizarse.

A continuación se presenta un procedimiento sencillo de seguir para lograr dicho fin.

(1) Mantener sujeto el cuatrirotor en todo momento durante la sintonización.
(2) Sintonizar primeramente el control de orientación solo. Definir R_d como la matriz identidad, $R_d = I_3$. Poner la ganancia derivativa en cero, $K_{D_R} = 0$. Ajustar la ganancia proporcional K_{P_R} hasta sentir que las acciones de control trabajan adecuadamente. Hacer oscilar manualmente el cuatrirotor para comprobar que no se desestabiliza.
(3) Ir aumentando la ganancia derivativa K_{D_R}. Hacer oscilar manualmente el cuatrirotor para comprobar que no se desestabiliza. Si el cuatrirotor se mantiene oscilando significa que nos hemos pasado del valor de esta ganancia.
(4) Sintonizar ahora el control de posición. Regresar la matriz de rotación deseada a su valor original, Eq. 10. Poner la ganancia derivativa en cero, $K_D = 0$. Ajustar la ganancia proporcional K_P hasta sentir que las acciones de control trabajan adecuadamente. Soltar el cuatrirotor con precaución, éste no debe desestabilizarse y debe realizar un vuelo suspendido con oscilaciones ligeras alrededor de la posición deseada, X_d.
(5) Ir aumentando la ganancia derivativa K_D. Soltar el cuatrirotor con precaución, éste no debe desestabilizarse y debe realizar un vuelo suspendido en la posición deseada, X_d. Si empieza a hacer oscilaciones bruscas significa que nos hemos pasado del valor de esta ganancia.

5. RESULTADOS EXPERIMENTALES

Con el fin de evaluar el funcionamiento del sistema y el desempeño del controlador propuestos, se realizó un vuelo experimental en condiciones controladas.

El experimento consiste en mantener un vuelo suspendido en la posición:

$$X_d = \begin{bmatrix} 0 \\ 0 \\ -0.4 \end{bmatrix} \tag{15}$$

esto es, en el centro del campo visible del OptiTrack y a una altura de 40cm. De acuerdo con la dinámica del cuatrirotor, esto implica que los ángulos de orientación ϕ y θ se deberán mantener en 0, mientras que el ángulo de guiñada deseado será $\psi_d = 0$.

En la Fig. 5 se muestra una gráfica de las posiciones medidas en los 3 ejes durante el experimento y la posición deseada en color rojo. Se puede observar que las posiciones son reguladas adecuadamente manteniendo un vuelo suspendido, con un error máximo de 10cm. Además, en esta misma gráfica, en el eje Z, se observa el despegue y aterrizaje del cuatrirotor.

En la Fig. 6 se muestra una gráfica de la orientación del cuatrirotor medida durante el experimeto. Se observa que la orientación tiene un error máximo de regulación de 6°.

Finalmente, en la Fig. 7 se muestra una gráfica en 3D de la posición del cuatrirotor durante el experimento.

En el enlace Exp (2018) se muestra un video del experimento realizado.

6. CONCLUSIONES

A principio de cuentas, parecería trivial el portar el código de un programa para el control de un cuatrirotor imple-

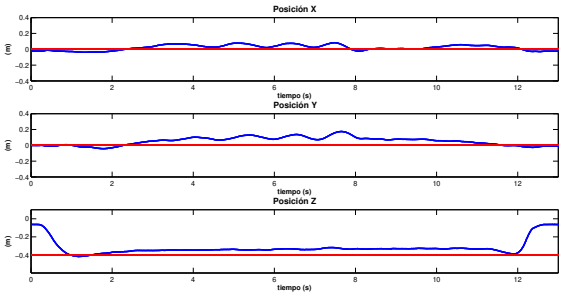

Fig. 5. Posición medida durante el experimento

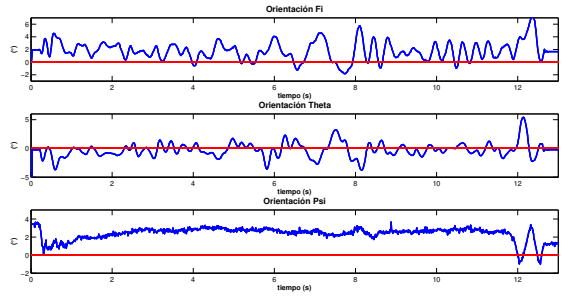

Fig. 6. Orientación medida durante el experimento

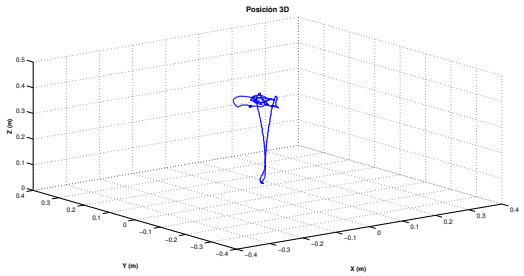

Fig. 7. Gráfica en 3D de las posiciones medidas

mentado en un MCU a un MPU, ya que son dispositivos muy similares. Sin embargo, se tiene que considerar que cuentan con arquitecturas diferentes y ambos fueron pensados para llevar a cabo tareas diferentes, ver Gaillard and Eieland (2013).

Un MCU cuenta con una mayor variedad de periféricos internos y con un limitado poder de procesamiento, mientras que un MPU se centra en tener el mayor poder de procesamiento posible dentro del integrado, lo que obliga a que ciertos periféricos, como la memoria RAM, se tengan que conectar de forma externa. Este mayor poder de procesamiento permite a los MPU ejecutar sistemas operativos para poder llevar a cabo tareas más complejas facilitando la interacción con el usuario.

En este artículo se abordaron los aspectos y consideraciones más importantes que se tienen que tomar en cuenta para la construcción de un cuatrirotor experimental controlado por un MPU.

En cuanto a la prueba experimental, es posible mejorar los errores de regulación haciendo una caracterización más exhaustiva de los motores, esto es, conocer el valor preciso de empuje que genera cada motor dependiendo de la señal PWM que se le aplique a cada uno de forma individual, ya que los motores a pesar de ser iguales, no son idénticos mecánicamente debido a los procesos de producción, y cualquier variación por mínima que sea en el empuje total de cuatrirotor generará un declive hacia cierto eje.

BIBLIOGRAFÍA

Erginer, B. and Altug, E. (2007). Modeling and pd control of a quadrotor vtol vehicle. In *2007 IEEE Intelligent Vehicles Symposium*, 894–899. doi:10.1109/IVS.2007.4290230.

Exp (2018). Raspberrypi-powered quadrotor hovering. URL `https://youtu.be/672YnEGcElA`.

Gaillard, F. and Eieland, A. (2013). Microprocessor (mpu) or microcontroller (mcu)? what factors should you consider when selecting the right processing device for your next design. URL `http://ww1.microchip.com/downloads/en/DeviceDoc/MCU_vs_MPU_Article.pdf`.

Garcia, A., Mattison, E., and Ghose, K. (2015). High-speed vision-based autonomous indoor navigation of a quadcopter. In *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, 338–347. doi:10.1109/ICUAS.2015.7152308.

Guadarrama-Olvera, J.R., Corona-Sánchez, J.J., and Rodríguez-Cortés, H. (2014). Hard real-time implementation of a nonlinear controller for the quadrotor helicopter. *Journal of Intelligent & Robotic Systems*, 73(1), 81–97. doi:10.1007/s10846-013-9962-z. URL `https://doi.org/10.1007/s10846-013-9962-z`.

Gupte, S., Mohandas, P.I.T., and Conrad, J.M. (2012). A survey of quadrotor unmanned aerial vehicles. In *2012 Proceedings of IEEE Southeastcon*, 1–6. doi:10.1109/SECon.2012.6196930.

Henzinger, T.A., Horowitz, B., and Kirsch, C.M. (2003). Giotto: a time-triggered language for embedded programming. *Proceedings of the IEEE*, 91(1), 84–99. doi:10.1109/JPROC.2002.805825.

Lynen, S., Achtelik, M.W., Weiss, S., Chli, M., and Siegwart, R. (2013). A robust and modular multi-sensor fusion approach applied to mav navigation. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3923–3929. doi:10.1109/IROS.2013.6696917.

Mian, A.A. and Daobo, W. (2008). Nonlinear flight control strategy for an underactuated quadrotor aerial robot. In *2008 IEEE International Conference on Networking, Sensing and Control*, 938–942. doi:10.1109/ICNSC.2008.4525351.

Tlatelpa-Osorio, Y. (2017). *Control para despegue y aterrizaje de un cuatrirotor en presencia de momentos y fuerzas externos*. Master's thesis, CINVESTAV.

Vásquez-Beltrán, M.A. and Rodríguez-Cortés, H. (2015). A total energy control system strategy for the quadrotor helicopter. In *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, 286–293. doi:10.1109/ICUAS.2015.7152302.

