Intérprete de Comandos para el Control de Vuelo de Drones

Rosendo M.J. Vargas Valle
Universidad Tecnológica Emiliano Zapata del Estado de Morelos
División Académica de Mecánica Industrial
Laboratorio de Internet de las Cosas
rosendovargas@utez.edu.mx

Jonathan Villanueva Tavira
Tecnológico Nacional de México/ Centro Nacional de Investigación y Desarrollo Tecnológico
Laboratorio de Sistemas Híbridos Inteligentes
Estudiante Posdoctoral
villanueva.jonathan@ieee.org

Andrea Magadán Salazar
Centro Nacional de Investigación y Desarrollo Tecnológico
Laboratorio de Inteligencia Artificial
Profesor -Investigador
magadan@cenidet.edu.mx

Miguel Ángel Basurto Pensado
Centro de Investigación en Ingeniería y Ciencias Aplicadas
Universidad Autónoma del Estado de Morelos
Morelos, Mexico
mbasurto@uaem.mx

Hector Miguel Buenabad Arias
Tecnológico Nacional de México/ Centro Nacional de Investigación y Desarrollo Tecnológico
Departamento de Ingeniería Mecánica
Morelos, Mexico
hsmigbuenabad@hotmail.com

Marilú Chávez Castillo
Universidad Tecnológica Emiliano Zapata del Estado de Morelos
División Académica de Mecánica Industrial
Profesor -Investigador
mariluchavez@utez.edu.mx

***Resumen*— En los últimos años los dispositivos autónomos han tomado mayor relevancia, como los drones, y cada vez es más común encontrarlos en la vida cotidiana, por lo tanto tenemos la necesidad de que estos sean más seguros para las personas que lo rodean. En este documento se describe un proyecto de investigación realizado en la Universidad Tecnológica Emiliano Zapata del Estado de Morelos, a fin de proponer una interfaz para el desarrollo de software de control de vuelo para dispositivos autónomos, como son los drones, realizando una propuesta de un intérprete de comandos que permite el control de vuelo de un drone cuadro rotor por la técnica de vuelo de navegación a estima.**

Se desarrolló y probó un prototipo para realizar una evaluación del desempeño en vuelo real con un mini drone de un peso aproximado de 500 gramos, evaluando la precisión y repetitividad de la ejecución de los programas desarrollados.

Palabras Claves: Drones, Vehículos Autónomos, interprete, IFR.

I. INTRODUCCIÓN

De acuerdo con datos de varias fuentes, incluyendo fuentes militares, se estima que no en poco tiempo la mayoría de las misiones aéreas de tipo militar serán realizadas por vehículos aéreos no tripulados [1]. Según datos de [1] el principal objetivo de reducir el error humano y reducir el riesgo para los pilotos aliados.

Un ejemplo de este tipo de proyectos es el vehículo autónomo de la empresa Boeing Little Bird [2], el cual se implementa en un helicóptero MH-6 ahora denominado AH-6, es capaz de despegue, vuelo y aterrizaje autónomo; pensado principalmente para el suministro autónomo de tropas, vigilancia y comunicaciones.

El logro máximo conocido, hasta ahora, es el vehículo autónomo orbital Boeing X37[3], inicia su desarrollo en 1999 por la NASA y Boeing, en 2004 se transfiere a la Darpa y es un proyecto clasificado desde entonces, sin embargo se sabe que rompió el record de vuelo en la órbita terrestre alcanzando 781 días [4], teniendo la capacidad de retornar al planeta y aterrizar en su base de forma autónoma.

Tomando en cuenta lo anterior, es innegable la importancia actual del desarrollo de vehículos autónomos también para usos civiles y capacitar a las futuras generaciones con los principios básicos de su diseño e implementación.

Un gran desafío es el desarrollo de software de control a alto nivel de estos dispositivos autónomos que logre la versatilidad necesaria para el desarrollo y modificación de rutas de vuelo de forma fácil y eficiente que pueda ser llevada a cabo por personal más relacionado con el desarrollo de rutas que en la programación de dispositivos.

II. INTÉRPRETE DE COMANDOS

Los intérpretes de comandos son una alternativa para el desarrollo de software porque tienen la capacidad de convertir instrucciones desde un texto a órdenes directas a un hardware o software específico, Muchos sistemas cuentan con al menos un intérprete de comandos [5]. Incluso lenguajes de programación, como java por ejemplo, utilizan esta técnica para facilitar la portabilidad de sus programas valiéndose de una máquina virtual la cual si tiene comunicación directa con el hardware de un equipo de cómputo [6]. En este proyecto en particular requerimos desarrollar una herramienta que facilite el desarrollo de secuencias de vuelo para un drone cuadro rotor Parrot ArDrone.

A. Parrot ArDrone

El Parrot ArDrone es un drone comercial desarrollado y comercializado por la empresa francesa Parrot con fines de entretenimiento presentado el año 2010 [7]. La particularidad principal de este drone es que desde su lanzamiento, no requiere de un control remoto especial pues se manipula desde un teléfono inteligente [7], adicionalmente el fabricante liberó un conjunto de librerías con las cuales se podrían desarrollar interfaces de control del drone desde una computadora con sistema operativo Linux que contara con una tarjeta de red inalámbrica [8].

Como trabajo previo, se desarrolló un intérprete de comandos en QtCreator C++ ejecutándose en Linux Kubuntu 12.1, este software funcionó correctamente pero tenía también problemas infranqueables, primero, el sistema tendía a bloqueos por problemas de compatibilidad de hardware en varios sistemas, y segundo, la imposibilidad de compilar nuevas aplicaciones con versiones más nuevas del núcleo de Linux como sucedió con la versión 13 de Kubuntu, todos estos problemas hicieron imposible continuar con su desarrollo con esa arquitectura. Sin embargo, tenía la ventaja de que el usuario no podía equivocarse al momento de escribir un programa pues todo se hacía a través de botones y controles numéricos, esto fue desarrollado así para evitar errores de codificación, además de generar el código maquina directamente (Ver Fig. 1).

Fig. 1. Primera versión del intérprete de comandos en QtCreator + AR.Drone SDK (Elaboración propia).

Afortunadamente muchos programadores desarrollaron sus propias versiones de estas librerías permitiendo su uso en IDEs de desarrollo como fue MATLAB y LabVIEW, en este último se decidió desarrollar una nueva versión de nuestro software dada su alta portabilidad, pues a pesar de nuevas actualizaciones del software LabVIEW, las librerías antiguas que datan de 2011 siguen ejecutándose sin ningún problema, convirtiendo al AR.Drone en una opción viable y barata para la prueba de algoritmos complejos avanzados (Ver Fig. 2).

Fig. 2. Interprete de comandos en LabVIEW (Elaboración propia).

Aunque existen varios Toolkits disponibles, se optó por utilizar la librería de “LVH AR Drone Toolkit” por ser de las más agiles y ligeras, compatible con todas las versiones de LabVIEW desde la versión 2011. Sin embargo, el uso de este Toolkit presenta varios inconvenientes, uno de los más importantes es que para la construcción de un comando se requieren los valores numéricos como consignas de valores angulares de los ejes de referencia del drone en 3 dimensiones (Ver Fig. 3).

Fig. 3. Ejes de referencia para las velocidades angulares del AR Drone [9].

En total se requieren 4 valores reales que van de 1 a -1, uno para cada uno de los ejes de referencia y otro más para la velocidad de rotación de las aspas para controlar la altura de vuelo del drone, estos datos deben integrarse en un clúster que conserve el orden correcto de las señales para evitar comportamientos erráticos de vuelo. El Parrot AR Drone está parcialmente automatizado y es capaz de realizar el despegue y aterrizaje de forma autónoma al indicárselo con una señal booleana, sin embargo como siempre está esperando una instrucción por parte del piloto, no cuenta con elementos propios para desarrollar un algoritmo para realizar una automatización más avanzada al menos que se realicen modificaciones a nivel hardware.

III. Intérprete de comandos de vuelo

Se requiere de un software que facilite el envío de comandos de vuelo al Drone por lo tanto que facilite su programación o la generación de comandos de vuelo, se opta por iniciar con un intérprete de comandos sencillo que reciba un archivo de texto con instrucciones simples de vuelo. Los comandos elegidos son: despegar, aterrizar, movimiento a la derecha, movimiento a la izquierda, girar a la derecha, girar a la izquierda, elevarse y descender. Estos comandos simples pueden representar de forma sencilla el movimiento del drone sobre los 3 ejes de las 3 dimensiones disponibles, además de ser fáciles de entender por los usuarios.

A. Identificacion de comandos

Un intérprete de comandos debe discriminar de forma automática a los comandos entre sí para poder realizar las acciones indicadas para cada comando, la forma más sencilla de hacer esto es definir un lenguaje o estructura de comandos para ser verificado carácter por carácter en una secuencia específica y así reconocer una palabra o comando. Para facilitar esto se decide reducir al mínimo los caracteres necesarios para cada uno de los comandos seleccionados quedando como sigue: Desp, ater, der, izq, rder, rizq, elev y desc.

Con esto se puede determinar inequívocamente a cada uno de los comandos con el menor número de caracteres, sin embargo, aunque con esto se puede determinar la dirección del movimiento, no es posible determinar la cantidad de movimiento a realizar. Una de las técnicas básicas de navegación es la navegación por estima, esta consiste en determinar la posición actual a través de cálculos que implican la dirección de desplazamiento y el tiempo transcurrido a determinada velocidad [10].

Esto último altera la sintaxis de nuestro sistema de comandos dejándolo como "*{comando}, {tiempo}*", como puede inferirse, falta la velocidad de vuelo, se tomó la decisión de tener una velocidad de vuelo constante durante la ejecución del vuelo, pero modificable e independiente en cada uno de los 3 ejes. Por último, se determina que deberá de existir un comando por línea de texto para facilitar la revisión de los mismos.

B. Verificacion de los programas

Es de vital importancia que los comandos ingresados sean correctos y coherentes para poder realizar una ejecución exitosa, para ello se desarrolla una rutina de verificación de código que sigue las siguientes reglas:

a) Los comandos deben estar escritos correctamente.

b) Todos los comandos deben estar acompañados de un parámetro de tiempo de duración del comando, a excepción del despegue y aterrizaje.

c) El parámetro de tiempo nunca debe ser cero.

d) Los comandos estarán separados del parámetro de tiempo por una coma.

e) El fin de línea será el fin de la instrucción, por lo tanto, un programa será verificado correctamente si se detectan la misma cantidad de comandos que de líneas en el documento de texto.

f) Una secuencia correcta deberá empezar siempre con un comando de despegue y finalizar siempre con un comando de aterrizaje.

IV. Implementacion

Como se indicó anteriormente este intérprete de comandos se implementó en LabVIEW para facilitar la experimentación de programas de automatización en el Parrot AR Drone en equipos de cómputo modernos. Por las características de LABVIEW se opta por una arquitectura de productor-consumidor.

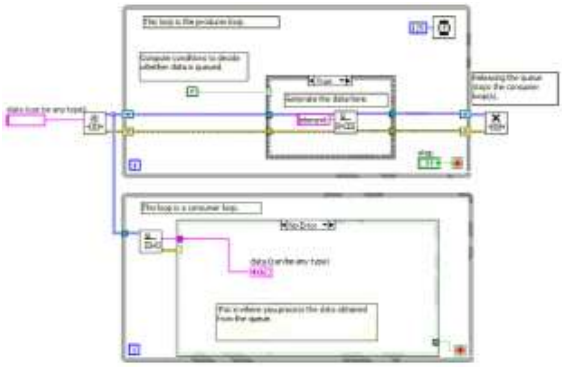

Fig. 4. Patrón de diseño productor/consumidor [11].

Donde el productor incluye una máquina de estados para identificar el comando a ejecutar y la cantidad de tiempo, enviando el comando identificado a un arreglo que funciona como entrada para el consumidor.

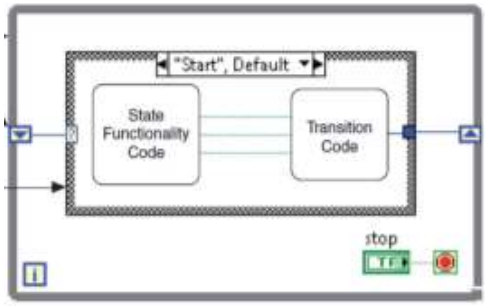

Fig. 5. Estructura básica de una máquina de estados [12].

El consumidor procesa el comando y genera un arreglo de valores que son los argumentos para producir una instrucción para el dron compatible con su librería de comunicación denominado comando AT que tiene el siguiente formato:

Corresponding API function : *ardrone_at_set_progress_cmd*
Corresponding API function : *ardrone_at_set_progress_cmd_with_magneto*

Syntax : AT*PCMD=%d,%d,%d,%d,%d,%d<CR>
Syntax : AT*PCMD_MAG=%d,%d,%d,%d,%d,%d,%d,%d,%d<CR>

Argument 1 : the sequence number
Argument 2 : flag enabling the use of progressive commands and/or the Combined Yaw mode (bitfield)
Argument 3 : drone left-right tilt - floating-point value in range [−1..1]
Argument 4 : drone front-back tilt - floating-point value in range [−1..1]
Argument 5 : drone vertical speed - floating-point value in range [−1..1]
Argument 6 : drone angular speed - floating-point value in range [−1..1]
Argument 7 : magneto psi (only for AT*PCMD_MAG) - floating-point value in range [−1..1]
Argument 8 : magneto psi accuracy (only for AT*PCMD_MAG) - floating-point value in range [−1..1]

Description :
This command controls the drone flight motions.

Fig. 6. Sintaxis de los comandos AT de movimiento de acuerdo con el ArDrone SDK [8].

Este comando es enviado por red wifi al dron para su ejecución. Fuera del ciclo de ejecución en una estructura adicional se coloca un paro de emergencia para que en caso de un funcionamiento inadecuado se detenga la ejecución del programa actual y se aterrice el dispositivo.

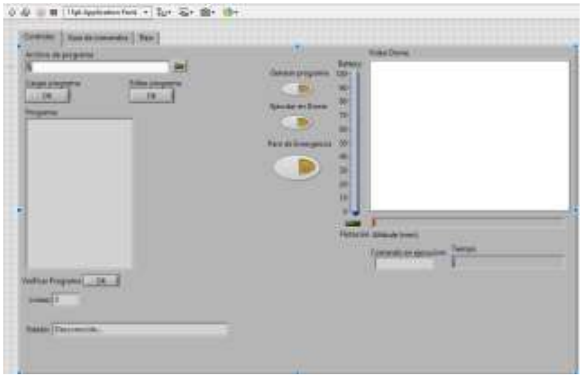

Fig. 7. Interfaz del sistema de interprete desarrollado (Elaboración propia).

Cabe mencionar que en las rutinas del SDK existe un paro de emergencia propio, sin embargo, este detiene por completo todo el funcionamiento del dron provocando que el mismo entre en caída libre, lo cual puede ser peligroso para los operadores y espectadores.

La interfaz cuenta con un visualizador de video capturado por la cámara del dron, un espacio donde se muestra el código a ejecuta, botones de verificación de código y ejecución, indicadores que muestran el comando ejecutado en tiempo real, así como el tiempo restante de ejecución de dicho comando con un gráfico de barras.

Se realizan una serie de pruebas para comprobar el funcionamiento. El protocolo de pruebas final queda como sigue:

a) Se escribe una rutina en un editor de textos simples de acuerdo a los comandos admitidos.
b) Se carga el código en el sistema.
c) Se ejecuta la verificación del código, descrito en III.B, esta produce el arreglo de instrucciones de vuelo.
d) Se realizan las verificaciones del estado del drone y del área segura para poder realizar el vuelo de prueba.
e) Se activa la ejecución de código oprimiendo el botón “Ejecutar en drone” para observar la ejecución de los comandos.
f) Se espera y observa la ejecución.
g) Se verifica que los movimientos sean correctos además de la repetitividad de las secuencias.

Al realizar la primera ronda de experimentos se hace evidente que se requería una ganancia variable para realizar movimientos más suaves pues anteriormente se utilizaba un valor constante de 1, por lo que se realiza la modificación.

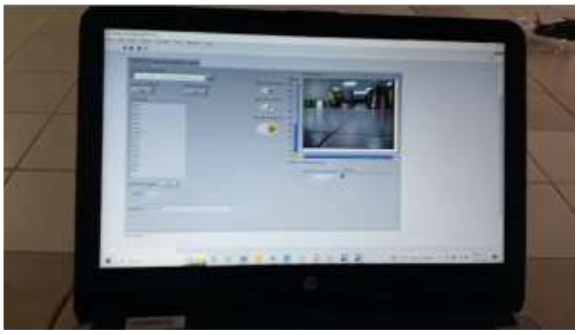

Fig. 8. Interfaz del sistema de intérprete en ejecución (Elaboración propia).

Se realizan varios experimentos posteriores siendo todos exitosos obteniendo los siguientes resultados.

Fig. 9. Dron en vuelo realizando ejecución (Elaboración propia).

V. Resultados

Las pruebas fueron exitosas, el prototipo realiza la ejecución de comandos de forma correcta y con alta repetitividad de movimientos en ambientes controlados, sin embargo se detectaron circunstancias en las que se pierde precisión y repetitividad.

Una situación en la que se pierde precisión es el vuelo en exteriores, por las limitaciones de nuestra base de pruebas, la cual pesa medio kilo, es sensible al viento natural, a pesar de contar con algoritmos de corrección, es inevitable que se tengan errores de precisión de vuelo proporcionales a la fuerza

provocada por cambio de la dirección del viento, que pueden ir desde unos centímetros hasta metros.

Los cambios bruscos de dirección afectan el desempeño de los movimientos dado que el dron debe estabilizarse antes de efectuar correctamente el cambio, afectando la duración efectiva del comando.

VI. Conclusiones

En el desarrollo de este prototipo se pudo concluir lo siguiente:

- El prototipo propuesto alcanza los objetivos planteados al inicio del proyecto de forma correcta y eficiente, principalmente en ambientes cerrados.
- La herramienta de software de trabajo elegida para este proyecto cumplió con las expectativas y resuelve los problemas de compatibilidad iniciales con el primer prototipo desarrollado.
- Aunque nuestro software cumple con el objetivo del correcto seguimiento de rutas preestablecidas, requiere de mejoras de precisión y seguridad en espacios abiertos.
- Aumentar el tamaño y peso del prototipo ayudará a resistir de mejor manera los cambios de dirección del viento.
- Nuestro prototipo de software es viable para el control de sistemas de transporte aéreo de mercancías y personas basadas en rutas preestablecidas como el transporte público.
- Esta técnica de planeación de vuelo es adecuada para sistemas que tengan problemas de comunicación con sistemas de ubicación como el GPS.

VII. Trabajos a futuro

Durante el desarrollo de las pruebas pudieron notarse posibles mejoras con respecto a su desempeño como se enlista a continuación:

- Es necesario implementar rutinas de vigilancia y en su caso de corrección de ruta, aunque el dispositivo cuenta con software de bajo nivel que las realiza, no son suficientes.
- Es importante implementar un sistema reactivo a objetos ajenos que se desplacen por la ruta, esto por un evento inesperado o errores en la ruta que esquiven o al menos pausen la ejecución de la ruta.
- Es recomendable buscar una solución para “suavizar” los cambios de dirección para mejorar la precisión de los movimientos, posiblemente con rutinas de estabilización propias a alto o bajo nivel.

Referencias

[1] U.S. Army UAS Center of excelence; “Eyes of the Army”; US Army Unmanned Aircraft Sistems Roadmap 2010-2035; Fort Rucker Alabama. Ed. 2010.

[2] Csaszar, Lawrence M.; “The Joint Tactical Aerial Resupply Vehicle Impact on Sustainment Operations”; US Army Command and General Staff College Fort Leavenworth United States; 2017.

[3] Blaesser, Nathaniel James. ” Mission Analysis and Conceptual Design of a Space Ambulance Utilizing the Boeing X-37B Platform” University of California, Davis ProQuest Dissertations Publishing, 2014. 1585047.

[4] Brett Tingley, “US military's mysterious X-37B space plane sets new spaceflight record”, SPACE.com, 2022; recuperado de: https://www.space.com/x-37b-space-plane-mission-duration-record

[5] Gómez, Ahijado, Flores, “Curso de Linux 2.0”, Universidad de los Andes, Colombia, 1999, en linea, recuperado de https://www.ibiblio.org/pub/linux/docs/LuCaS/Tutoriales/CURSOLINUX/curso_linux/curso_linux.html

[6] Gracia A. “Curso de lenguaje java”, Universidad del pais Vasco, españa, 2000, en linea, recuperado de http://www.sc.ehu.es/sbweb/fisica/curso.htm.

[7] Parrot, “AR.Drone User Guide”, 2010, en linea recuperado de https://www.parrot.com/assets/s3fs-public/2021-09/ar-drone_user-guide_uk.pdf

[8] Piskorki, Brulez, Eline, Dhayer, “AR.Drone Developers Guide”, 2012, en linea, recuperado de https://jpchanson.github.io/ARdrone/ParrotDevGuide.pdf

[9] Silva, Peña, Mendoza, “Sistema de inspección y vigilancia utilizando un robot aéreo guiado mediante visión artificial”, ITEKNE Vol. 10, 2013, ISSN: 1692-1798.

[10] Muñoz Miguel, “Teroria basica para aviones ligeros”, 2021, documento en linea recuperado de https://www.manualvuelo.es/index.html

[11] National Instrument, “LabView Core 2”, manual de curso de certificacion, 2012.

[12] National Instrument, “LabView Core 1”, manual de curso de certificacion, 2012.