UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería

MEGAPROYECTO SEEQ ROBOHELICÓPTERO

Trabajo de investigación presentado por los siguientes estudiantes para optar al grado académico de Licenciatura en Ingeniería Mecatrónica:

Mario Andrés Búrbano Castro
José Eduardo Morales Espinoza
Melisa Sandoval Mayén
Roberto Enrique Saravia Fernández
Daniel Alejandro Suazo Salazar

Guatemala,

2011

MEGAPROYECTO SEEQ ROBOHELICÓPTERO

UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería

MEGAPROYECTO SEEQ ROBOHELICÓPTERO:

Trabajo de investigación presentado por los siguientes estudiantes para optar al grado académico de Licenciatura en Ingeniería Mecatrónica:

Mario Andrés Búrbano Castro

José Eduardo Morales Espinoza

Melisa Sandoval Mayén

Roberto Enrique Saravia Fernández

Daniel Alejandro Suazo Salazar

Guatemala,

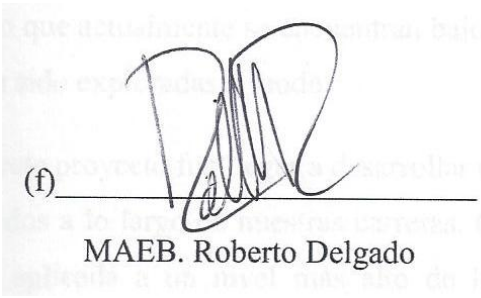
2011

Vo.Bo.:

(f)_____

MAEB. Otto Girón

Tribunal:



(f)_____

MAEB. Roberto Delgado

Fecha de presentación: 29 de noviembre de 2011

PREFACIO

El megaproyecto SEEQ Robohelicóptero surge por la motivación de abrir una nueva línea de investigación en la Universidad del Valle de Guatemala. El nombre del proyecto, corresponde a sus siglas en inglés "*Smart Environment Explorer Quadcopter*", el cual describe de manera acertada sus objetivos. El proyecto involucra distintas ramas de conocimiento científico que actualmente se encuentran bajo desarrollo en otros países, pero en Guatemala no han sido exploradas a fondo.

La inspiración de este proyecto fue llegar a desarrollar tecnología nueva aplicando los conocimientos adquiridos a lo largo de nuestras carreras. Generar investigación y un precedente en ingeniería aplicada a un nivel más alto de lo convencional. Se buscó innovar y demostrar que en el país hay capacidad para contribuir en temas recientes e importantes a nivel mundial.

El trabajo llevado a cabo durante todo el proceso involucró un gran esfuerzo y dedicación por parte de los integrantes del grupo. Este esfuerzo dio resultados positivos, y esperamos que los mismos sirvan de inspiración para futuras investigaciones. Pensamos en este proyecto como una primera etapa de un prometedor futuro en el área.

Se desea agradecer a todas las personas que contribuyeron a que el proyecto se llevara a cabo con éxito. A nuestras familias, por todo el apoyo y la paciencia mostrados durante el desarrollo del trabajo. A nuestros asesores MAEB. Otto Girón y MAEB. Roberto Delgado por sus aportes y su guía, y a todas las personas que de manera directa o indirecta se involucraron con el proyecto.

ÍNDICE

Página

ÍNDICE DE FIGURAS

Página

ÍNDICE DE TABLAS

Página

RESUMEN

El nombre SEEQ corresponde a las palabras en inglés “Smart Environment Explorer Quadcopter”. Este proyecto consistió en diseñar y construir un robot aéreo que ejecuta su vuelo y movimientos mediante cuatro rotores. Se le llama inteligente, ya que el robot genera un mapa del ambiente inexplorado en el que se encuentra, utilizando los datos que se obtienen de un sensor láser. El mapa generado lo utiliza el robot para estabilizarse, localizarse, y navegar dentro del ambiente en el que se encuentra. Estos mapas generados se transmiten a la etapa de visualización, en donde se muestra un mapa tridimensional y uno de planta del lugar. Esta interfaz final le permite al usuario movilizarse en el ambiente tridimensional.

El objetivo principal del proyecto fue desarrollar un vehículo que domine el medio aéreo, con fines de exploración, generación y visualización de mapas en ambientes desconocidos. Este objetivo se logró, dividiendo el proyecto en cinco módulos: estructura, estabilización y control, sensor e interfaz R2-232, mapeo y exploración y visualización 3D.

En el módulo de la estructura, se contemplan los aspectos relacionados con el diseño de una estructura robusta, considerando análisis de esfuerzos, vibraciones y deflexiones. Se realiza una selección de los motores y hélices apropiados que generen la sustentación necesaria para elevar cuatro kilogramos. Se lleva a cabo un control local frente a obstáculos que el control de posición no pueda manejar en caso de una pérdida de comunicación. Finalmente se caracteriza el sistema y se obtienen las funciones de transferencia para el control de estabilidad.

En el módulo de estabilización y control se trabajó la estimación de los ángulos de inclinación del helicóptero y su control. Se crea un control de ángulos de inclinación, y comandos de control externos vía RF para controlarlo.

En el módulo de sensor e interfaz RS-232, se encargó obtener los datos del ambiente en el que se encuentra, comunicárselos vía USB a un micro controlador y de enviárselos vía radio frecuencia al módulo donde se realiza el mapeo y la exploración.

El módulo de mapeo y exploración, se enfoca en la elaboración de la lógica interna que permite al robot realizar un vuelo de manera autónoma. En este módulo se obtienen los datos del sensor láser continuamente, por lo que es necesario que se haga un análisis de los mismos, y que continuamente se agreguen a un mapa global en la posición donde corresponden. Este mapa es de dos dimensiones y es muy importante, ya que retroalimenta al helicóptero con su posición dentro del mapa y además lo ayuda a estabilizarse. Este mapa también se utiliza para escoger la mejor posición a la que debería moverse el helicóptero para encontrar más datos útiles y además se utiliza en el módulo de visualización para generar el mapa con el cual el usuario puede interactuar.

El módulo de visualización se encarga de tomar el mapa generado por el módulo de mapeo y exploración y a cada punto le asigna una altura específica para que el usuario pueda tener una representación muy cercana a la realidad del ambiente en donde navega el helicóptero Dentro de este mapa el usuario se puede movilizar para explorar todas las partes del mapa que se hayan graficado en la aplicación. Además en el programa se tiene una visualización de planta, en donde se puede ver la posición en donde está explorando el mapa el usuario y hacia donde se encuentra viendo.

Una de las limitantes para la solución planteada en éste trabajo, es que el ambiente a explorar debe ser un ambiente interno. Además se requiere que la rotación del robot con respecto a los tres ejes X, Y y Z sea limitada, y que cambie a una velocidad baja.

I. INTRODUCCIÓN

Existen muchas áreas de investigación en el campo de la ingeniería. El objetivo de estas investigaciones es proponer soluciones a problemas que facilitarán y mejoraran muchas de las actividades que se llevan a cabo en la vida cotidiana y la industria.

Uno de los elementos importantes de la vida cotidiana reciente, son los vehículos aéreos. Los vehículos aéreos existen desde varias décadas atrás, sin embargo, la mayoría necesita de alguna persona, para que los controle y los movilice dentro de algún ambiente. En los últimos años, han existido distintos proyectos en donde un robot aprende a reconocer el ambiente en donde se encuentra y se moviliza dentro del mismo. Esta habilidad de reconocer ambientes y movilizarse de manera autónoma, tiene sus orígenes en uno de los problemas científicos que más interés ha generado en la actualidad. Este es el problema de localización y generación de mapas simultáneamente, o SLAM por sus siglas en inglés. El problema SLAM plantea la pregunta: ¿Es posible para un vehículo autónomo empezar en una localización desconocida en un ambiente desconocido, e incrementalmente construir un mapa de este ambiente mientras se usa el mismo para computar la posición absoluta del vehículo? [38].

Esta línea de investigación ya desarrollada a nivel internacional, pero inexistente en el país, sirvió como inspiración para el megaproyecto SEEQ.

En el megaproyecto SEEQ, se pretende realizar un robot aéreo que no necesite la ayuda de una persona para estabilizarse y movilizarse dentro de un ambiente; además se pretende que el robot sea capaz de explorar su entorno e incluso retroalimentar a alguna persona de la forma del lugar en el que se encuentra.

Específicamente se realizó un robo helicóptero a base de fibra de vidrio, debido a las propiedades de resistencia y peso que este material ofrece. Se utilizaron cuatro motores para manejar cada rotor incorporado en el mismo y se utilizó una batería de 5 amperios hora. Para la estabilización se utilizó un acelerómetro para controlar en todo

momento la posición en el espacio del helicóptero. Este control fue bastante complicado, ya que el sensor utilizado devuelve bastante ruido en la señal de interés.

La siguiente etapa en el proyecto fue realizar el mapeo del entorno. Para esto se utilizó un sensor láser el cual obtiene datos de objetos cercanos al helicóptero y los envía de forma inalámbrica al módulo de mapeo y exploración, el cual ejecuta sus algoritmos en una computadora. Este módulo analiza los datos y los incorpora a un mapa global. El mapa global es un elemento muy importante para el proyecto, ya que este se utiliza para retroalimentar al helicóptero y mejorar la estabilización del mismo.

En la última etapa, el mapa global creado se envía al módulo de visualización en 3D. En este módulo se genera un mapa en tres dimensiones que ayuda al usuario a comprender los datos que está recopilando el helicóptero. Este mapa se actualiza en tiempo real, y se realiza por medio de internet, por lo que se puede abrir la aplicación en cualquier lugar del mundo, y se pueden observar los datos del mapa en el que se encuentra el robot.

Por medio de la unificación de todos los módulos mencionados, se llegó al resultado final de SEEQ, y se logró alcanzar los objetivos propuestos. Se logró obtener un robo helicóptero el cual es capaz de construir una representación de cualquier ambiente interior de manera autónoma, y a través de esta representación se sabe ubicar en el ambiente y evadir obstáculos. Se obtuvo un helicóptero con un vuelo bastante estable y con una capacidad de movimiento hacia donde se le indica bastante buena. Aunque no se llenaron por completo las expectativas, se entiende que este proyecto esta categorizado para un nivel de estudio más alto, por lo que los resultados del megaproyecto SEEQ fueron bastante buenos. Se consideran los resultados como una primera etapa en esta línea de investigación.

II. OBJETIVOS

A. OBJETIVO GENERAL

- Desarrollar un vehículo que domine el medio aéreo con fines de exploración, generación y visualización de mapas en ambientes desconocidos.

B. OBJETIVOS ESPECÍFICOS

- Diseñar e implementar una estructura aérea robusta y duradera capaz de levantar una masa de cuatro kilogramos con un circuito local de emergencia ante obstáculos cercanos.
- Diseñar y construir un explorador aéreo estabilizado electrónicamente y controlado mediante instrucciones sencillas.
- Implementar una interfaz de datos inalámbrica para el sensor laser de distancia Hokuyo URG-04LX-UG01 basada en el estándar RS-232 que provea una forma sencilla y eficiente de utilizarlo en aplicaciones robóticas.
- Implementar un algoritmo de mapeo y localización simultanea combinado con una estrategia de exploración que le permita al helicóptero realizar un vuelo de manera autónoma.
- Realizar un modelo tridimensional en tiempo real que sea representativo del lugar que se está explorando.

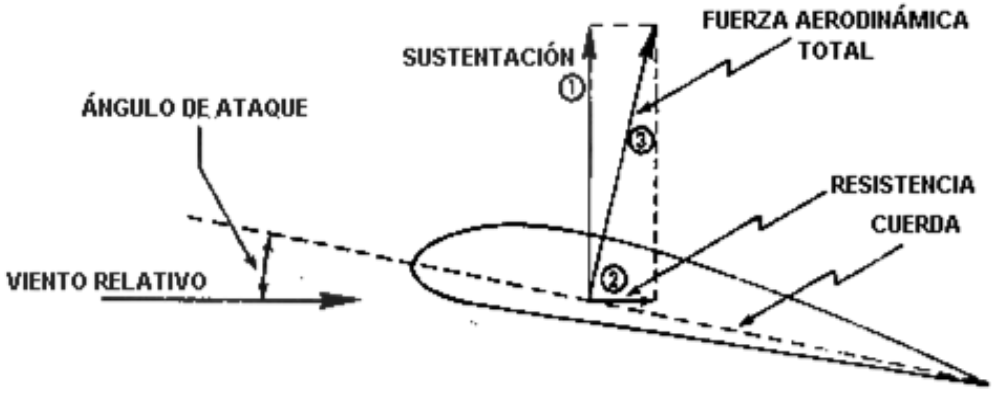
III. MARCO TEÓRICO

A. ESTRUCTURA

1. MECÁNICA DE FLUIDOS PARA AERODINÁMICA

a. Fuerza aerodinámica. La fuerza aerodinámica es la fuerza que se genera cuando un fluido pasa por una superficie a cierta velocidad. Cuando dicho fluido choca contra el perfil aerodinámico cierta parte se desplaza sobre éste y cierta parte se desplaza por debajo. La parte que se va por la parte superior se desplaza con una mayor velocidad a una menor presión y la parte que se va por la parte inferior se desplaza a menor velocidad y a mayor presión. Esta diferencia de presión en ambos lados del perfil, crean una fuerza resultante cuyas proyecciones vertical y horizontal se conocen como sustentación y arrastre respectivamente. En la Figura No. 1 se observa un diagrama donde se muestran la fuerza de neta aerodinámica, la fuerza de sustentación y la fuerza de arrastre. [35]

FIGURA NO. 1. MODELO DE FUERZA AERODINÁMICA, SUSTENTACIÓN Y ARRASTRE SOBRE UN PERFIL. [35]



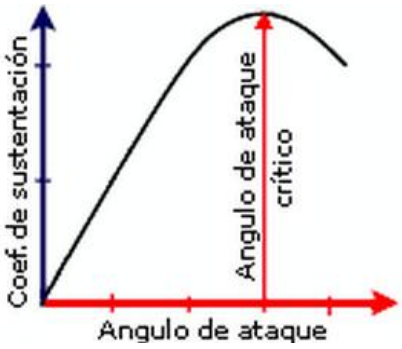
b. Fuerza de sustentación. La fuerza de sustentación es la componente vertical de la fuerza aerodinámica perpendicular al flujo del fluido, dicha fuerza es la responsable de crear el empuje para acelerar una masa verticalmente venciendo la fuerza de la gravedad. El modelo utilizado para determinar la magnitud de dicha fuerza es el siguiente.

FIGURA NO. 2. FUERZA DE SUSTENTACIÓN

$$L = \frac{1}{2} * \delta * Cl * A * V^2$$

Donde A es el área superficial del ala, δ es la densidad del fluido circundante, Cl es el coeficiente de sustentación y V es la velocidad del fluido en relación al ala. En general esta ecuación es compleja de determinar teóricamente. Los valores de densidad, área del perfil aerodinámico y la velocidad son variables fácilmente calculables teóricamente, sin embargo, el problema radica en el coeficiente de sustentación, normalmente los fabricantes de hélices proporcionan dichos coeficientes para los perfiles que ellos fabrican. Si un diseñador de un sistema aerodinámico no posee los datos del fabricante, debería ser capaz de calcular el coeficiente de sustentación a partir de modelos teóricos. El coeficiente de sustentación es básicamente una función que depende del ángulo de inclinación y el número de Reynolds. El comportamiento que presenta un coeficiente de sustentación Cl con relación al ángulo de inclinación a un valor de velocidad específica, es lineal hasta un ángulo óptimo, luego de este ángulo la sustentación disminuye. En la Figura No. 3 se muestra una curva particular para un perfil aerodinámico. [35]

FIGURA NO. 3. COEFICIENTE DE SUSTENTACIÓN VERSUS ÁNGULO DE ATAQUE. [36]



Normalmente esta teoría aplica para perfiles de alas de aviones, sin embargo, también es válida para las hélices en el caso de helicópteros. En este último caso el área y la velocidad varía con el radio. La idea en una hélice es poder generar una sustentación uniforme variando el ángulo de inclinación a lo largo de ésta, de manera que como la velocidad lineal aumenta con el radio, el ángulo disminuya con el radio.

c. Fuerza de arrastre. Cuando se habla de fuerza de arrastre se refiere a la componente de la fuerza aerodinámica que se encuentra paralela al flujo de aire que atraviesa el perfil. El arrastre total sobre un cuerpo se debe a dos componentes: el arrastre por fricción y el arrastre de presión o de forma. El modelo matemático del arrastre es similar al de la sustentación con la diferencia que en lugar del coeficiente de sustentación Cl se utiliza el coeficiente de arrastre Cd. El modelo entonces es el siguiente.

FIGURA NO. 4. FUERZA DE ARRASTRE

$$\mathrm{Fd} = \frac{1}{2} * \delta * \mathrm{Cd} * \mathrm{A} * \mathrm{V}^2$$

Al igual que el modelo utilizado para la sustentación, el coeficiente Cd es un coeficiente que el fabricante proporciona para los perfiles particulares que éste distribuye.

2. CUADCÓPTERO, FUERZAS Y REGLAS DE ESTABILIZACIÓN.

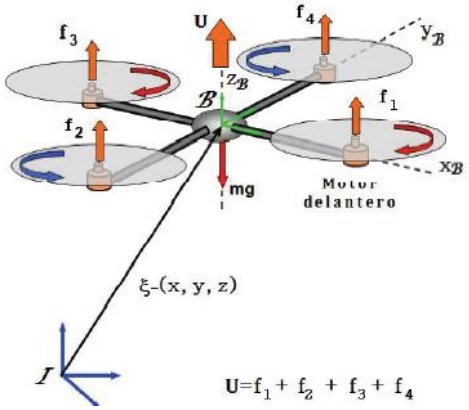
Un cuadcóptero es un vehículo aéreo formado por cuatro hélices adaptadas a cuatro rotores. Cuando todas las hélices comienzan a girar crean la sustentación necesaria para elevarlo. Dado que este no es un sistema estable presentará seis grados de libertad: movimiento vertical, movimiento horizontal (en dos direcciones), inclinación (en dos direcciones) y rotación. En la Figura No. 5 se muestra un cuadcóptero real.

FIGURA NO. 5. MODELO DE UN CUADCÓPTERO. [17]

Para poder comprender de una forma intuitiva los movimientos del cuadcóptero, es necesario determinar un diagrama de las fuerzas que interactúan en el vuelo, esto es un diagrama de cuerpo libre tridimensional.

Existen básicamente dos efectos que se producen cuando un rotor con hélice gira, en la sección anterior se describieron pero desde el punto de vista de la mecánica de fluidos, la sustentación se expresó como aquella componente que es perpendicular y el arrastre como aquella fuerza que es paralela al flujo de aire cortado por una superficie aerodinámica. En el caso de un cuadcóptero el flujo es cortado por medio de una hélice, esto quiere decir que la sustentación será entonces una fuerza vertical y el arrastre será una fuerza horizontal que se opone al movimiento de la hélice. La sustentación crea una aceleración ascendente y el arrastre crea un torque en sentido contrario al del giro del motor. Existe además la fuerza del peso, esta fuerza es la que atrae a la masa hacia el suelo y se localiza en el centro de masa del cuadcóptero. En la Figura No. 6 se muestran las fuerzas y momentos. [9]

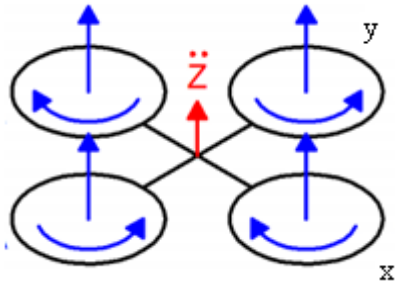
FIGURA NO. 6. SUMATORIA DE FUERZAS QUE ACTÚAN SOBRE UN CUADCÓPTERO.



En la Figura No. 6 se puede notar que dos rotores deben girar hacia el mismo y los otros dos motores hacia el otro sentido para evitar la rotación sobre el eje vertical al permitir que la sumatoria de torques sea cancelada. Antes de continuar con las reglas de estabilización se expondrán los posibles movimientos sobre un cuadcóptero y el porqué de los mismos.

El primer tipo de movimiento quizá sea el más sencillo e intuitivo de observar, se trata del movimiento vertical (*throttle*). Como ya se ha mencionado con anterioridad la fuerza de sustentación logra crear una aceleración vertical, de manera que combinada con el peso permiten que se realice un desplazamiento en sentido positivo o en sentido negativo sobre el eje z. En la Figura No. 7 se puede observar dicho comportamiento. [9]

FIGURA NO. 7. VERTICAL SOBRE EL EJE Z DE UN CUADCÓPTERO. [9]



El segundo tipo de movimiento es el movimiento de inclinación, este se produce tanto sobre el eje x (Roll) como sobre el eje y (pitch). Estos movimientos se deben a las dos fuerzas que se generan sobre los ejes y por consiguiente un torque aplicado en el centro de masa del cuerpo en movimiento. Si ambos torques son iguales permanece en

estado estacionario. Si un torque es mayor que al otro, se produce un giro positivo o negativo dependiendo de cuál sea el caso. Si se desea que el helicóptero no presente este tipo de movimiento, los torques pueden ser cancelados modificando el nivel de sustentación de los motores o modificando el brazo desde el centro de masa hasta el punto de colocación del motor. En la Figura No. 8 y Figura No. 9 se puede observar este tipo de movimiento. [9]

FIGURA NO. 8. INCLINACIÓN SOBRE EL EJE X DE UN CUADCÓPTERO. [9]

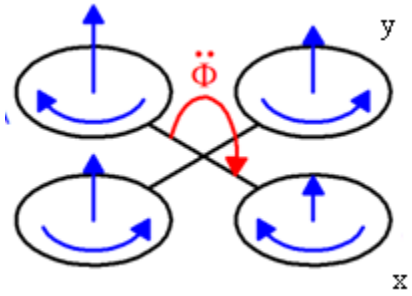
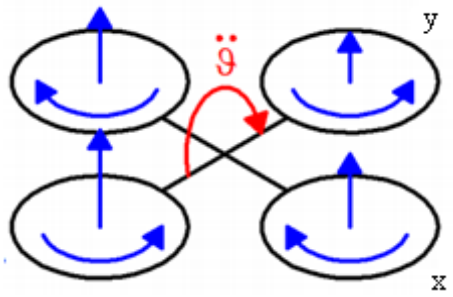


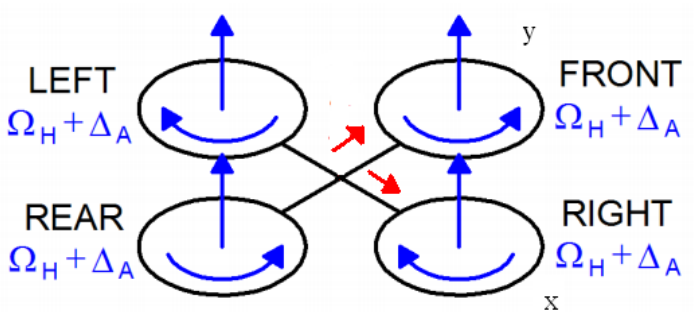
FIGURA NO. 9. INCLINACIÓN SOBRE EL EJE Y DE UN CUADCÓPTERO. [9]



El tercer tipo de movimiento es el de desplazamiento lineal sobre el eje X, y el desplazamiento lineal sobre el eje Y. Como se dijo en el movimiento anterior, el cuadcóptero puede inclinarse ya sea sobre el eje X o sobre el eje Y, cuando esto ocurre, el vector de sustentación es también rotado y aparece una proyección de esta fuerza sobre el eje sobre el cual ocurrió la rotación, ya sea en sentido negativo o positivo. Si se trata de un giro sobre el eje X la sustentación sobre el eje Y es proyectada y viceversa. Si se asume que el eje Y es el eje correspondiente al frente del cuadcóptero, este podrá avanzar o retroceder cuando haya rotación sobre el eje X, y además podrá moverse hacia los lados

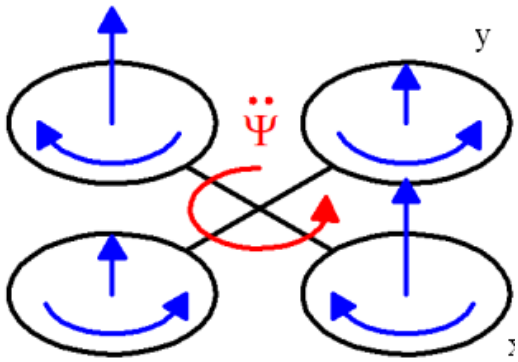
cuando exista rotación sobre el eje Y. Lo anteriormente explicado se puede observar en la Figura No. 10. [9]

FIGURA NO. 10. MOVIMIENTOS DE UN CUADCÓPTERO. [9]



Finalmente, el último tipo de movimiento es el movimiento de rotación sobre el eje vertical (Yaw). Este movimiento es producto del arrastre que existe en los motores, dado que dos motores giran en sentido horario y los restantes en sentido anti-horario, es posible crear un movimiento de rotación cuando se incrementa la velocidad en dos motores que giran en el mismo sentido, creando así una aceleración angular. [9]

FIGURA NO. 11. ROTACIÓN SOBRE EL EJE Z DE UN CUADCÓPTERO. [9]



Las fuerzas y momentos que afectan al sistema de un cuadcóptero puede modelarse con la sumatoria de fuerzas y momentos con respecto a algún punto. Estas condiciones se deberán cumplir en los seis ejes de movimiento. Expresado matemáticamente con la convención de símbolos expresados en las figuras anteriores, se obtiene lo siguiente.

FIGURA NO. 12. ECUACIONES DE MOVIMIENTO DEL CUADCÓPTERO

$$\Sigma Tx = Ixx * \frac{d^2\vartheta}{dt^2}$$

$$\Sigma Ty = Iyy * \frac{d^2\Phi}{dt^2}$$

$$\Sigma Tz = Izz * \frac{d^2\Psi}{dt^2}$$

$$\Sigma Fx = m * \frac{d^2x}{dt^2}$$

$$\Sigma Fy = m * \frac{d^2y}{dt^2}$$

$$\Sigma Fz = m * \frac{d^2z}{dt^2}$$

En el caso de estabilidad del cuadcóptero se deberá hacer que todas estas ecuaciones se igualen a cero. La estabilización se puede realizar mecánicamente, pero con mucha dificultad y fácil desajuste o por medio de un sistema de control digital obteniendo un sistema más robusto. [6]

3. VIBRACIONES MECÁNICAS. Todos los materiales se pueden modelar como un resorte. De hecho se podrían modelar como un resorte más amortiguador debido que en los materiales siempre existe un componente de amortiguamiento que disipa energía. Sin embargo, este componente es pequeño para muchas configuraciones, y se hace necesario la colocación de un amortiguador, cuando se requiere que las oscilaciones disminuyan con el tiempo. [5]

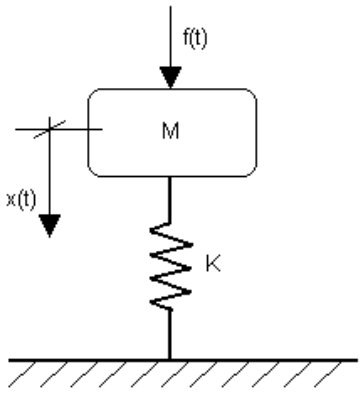
Cuando el sistema es libre de amortiguamiento, es decir, simplemente un resorte con una masa como el de la Figura No. 14, la ecuación diferencial que modela su movimiento es la siguiente.

FIGURA NO. 13. ECUACIÓN SISTEMA VIBRATORIO NO AMORTIGUADO

$$\frac{d^2x}{dt^2} + \frac{k}{m}x = \frac{f(t)}{m}$$

Donde k es la constante de resorte, m es la masa que oscila, x es el desplazamiento en función del tiempo y $f(t)$ es la fuerza externa que mueve la masa. [69]

FIGURA NO. 14. MODELO DE SISTEMA VIBRATORIO SIN AMORTIGUAMIENTO.



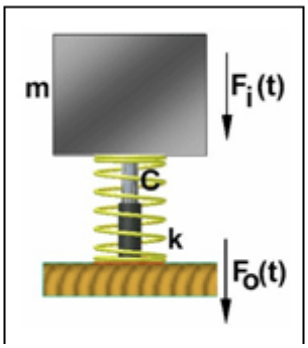
Cuando el sistema no es libre de amortiguamiento, es decir, la unión de un resorte y un amortiguador como en de la Figura No. 16, la ecuación diferencial que modela su movimiento es la siguiente.

FIGURA NO. 15. ECUACIÒN DE SISTEMA VIBRATORIO AMORTIGUADO.

$$\frac{d^2x}{dt^2} + \frac{c}{m}\frac{dx}{dt} + \frac{k}{m}x = \frac{f(t)}{m}$$

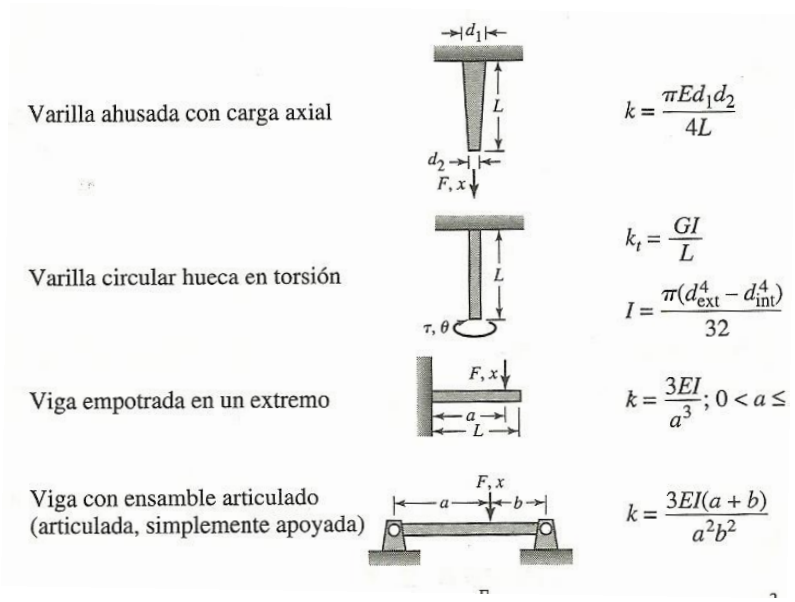
Donde k es la constante de resorte, c es la constante de amortiguamiento, m es la masa oscilante, x es el desplazamiento en función del tiempo y $f(t)$ es la fuerza externa que mueve la masa. [69]

FIGURA NO. 16. MODELO DE SISTEMA VIBRATORIO CON AMORTIGUAMIENTO. [35]



Muchos de los sistemas reales se comportan como resortes, por ejemplo, una bolla dentro de un fluido, un embolo que comprime un gas, un péndulo, una viga que sostiene una masa en su extremo, etc. Para muchos sistemas mecánicos el interés particular se da sobre los elementos que se puedan modelar como vigas en diferentes configuraciones. La Figura No. 17 muestra las constantes de resortes equivalentes para ciertas configuraciones de vigas a diferentes configuraciones de cargas. [5]

FIGURA NO. 17. CONSTANTES DE RESORTE PARA ALGUNOS ELEMENTOS ELÁSTICOS COMUNES. [5]



En maquinaria las vibraciones normalmente son producidas por los motores. Un motor es un sistema que debe producir movimiento circular, este movimiento tiende a crear aceleración centrípeta a lo largo del radio del elemento giratorio, si toda la masa estuviese perfectamente distribuida, la aceleración centrípeta resultante se cancelaría, sin

embargo, no es posible lograr un balance perfecto en motores ni en los elementos giratorios, como por ejemplo, las hélices.

Un motor desbalanceado sobre un cuerpo es posible modelarlo como una masa sujeta a un resorte y a un amortiguador. La solución de la ecuación diferencial de este sistema cuando la fuerza de excitación es de la forma $f(t) = F_o sen(\omega t)u(t)$ es la siguiente:

FIGURA NO. 18. ECUACIÓN DE DESPLAZAMIENTO PARA UN SISTEMA VIBRATORIO AMORTIGUADO

$$x(\tau) = \frac{Fo}{k} H(\Omega) sen(\left(\Omega\tau - \theta(\tau)\right) + \frac{Fo\, H(\Omega)\Omega\, e^{-\xi\tau}\, sen(\tau\sqrt{1-\xi^2} + \theta t(\Omega)}{k\sqrt{1-\xi^2}}$$

Donde

$$H(\Omega) = \frac{1}{\sqrt{(1-\Omega^2)^2 + (2\xi\Omega)^2}}$$

$$\theta(\tau) = \tan^{-1}\frac{2\xi\Omega}{1-\Omega^2}$$

$$\theta t(\tau) = \tan^{-1}\frac{2\xi\sqrt{1-\xi^2}}{2\xi^2 - (1-\Omega^2)}$$

$$\xi = \frac{c\omega n}{2k}$$

$$\omega n = \sqrt{\frac{k}{m}}$$

$$\tau = \omega n * t$$

$$\Omega = \frac{\omega}{\omega n}$$

De este resultado es importante notar que la variable ωn es la frecuencia natural del sistema, es decir que a esta frecuencia oscilaría el sistema simplemente dando un impulso. La variable τ es una constante de tiempo adimensional. La variable Ω es una relación entre la frecuencia de excitación y la frecuencia natural del sistema, Fo es la

magnitud de la fuerza oscilatoria de excitación, k es la constante de resorte equivalente del sistema, c es la constante de amortiguación equivalente del sistema y m es la masa que oscila. [5]

El resultado de la ecuación diferencial para el movimiento de la masa oscilante, está compuesta por dos partes la primera se conoce como régimen permanente y la segunda como régimen transitorio. En general para vibraciones mecánicas es de mayor interés el régimen permanente ya que éste muestra como es el movimiento durante tiempos prolongados cuando la maquinaria ya está funcionando o ha estado funcionando. La solución entonces tomando solamente el régimen permanente queda de la siguiente manera. [5]

FIGURA NO. 19. SOLUCIÓN EN RÉGIMEN PERMANENTE DE ECUACIÓN DE DESPLAZAMIENTO

$$x(\tau) = \frac{Fo}{k} H(\Omega)\text{sen}(\left(\Omega\tau - \theta(\tau)\right)$$

En el caso particular de una fuerza oscilatoria producida por desbalance en hélices o rotores, la fuerza se expresa como de la siguiente forma.

FIGURA NO. 20. FUERZA DE DESBALANCE EN UN ROTOR

$$Fa = mo * \epsilon * \omega^2$$

Donde mo representa la masa de desbalance, ϵ el radio hacia donde se encuentra esta masa y ω la velocidad angular del motor. Además se debe notar que el término m es igual a la masa del rotor más la pequeña masa de desbalance que en muchas ocasiones se desprecia. La función de movimiento para una masa de desbalance producida por un rotor se expresa de la siguiente manera. [5]

FIGURA NO. 21. ECUACIÓN DE VIBRACIÓN EN UN ROTOR DESBALANCEADO

$$x(\tau) = M\epsilon * \frac{\Omega^2}{\sqrt{(1-\Omega^2)^2 + (2\xi\Omega)^2}} * \text{sen}(\left(\Omega\tau - \theta(\tau)\right)$$

Donde

$$M\epsilon = \frac{mo * \epsilon}{m}$$

Dado que la vibración es algo que siempre ocurrirá mientras existan motores, el diseñador deberá establecer un sistema en donde la vibración esté reducida a valores aceptables para la aplicación que se esté implementando. Cuando no existe amortiguamiento o cuando este es muy pequeño, se puede decir que $\xi = 0$ y la ecuación anterior se reduce a la siguiente expresión. [5]

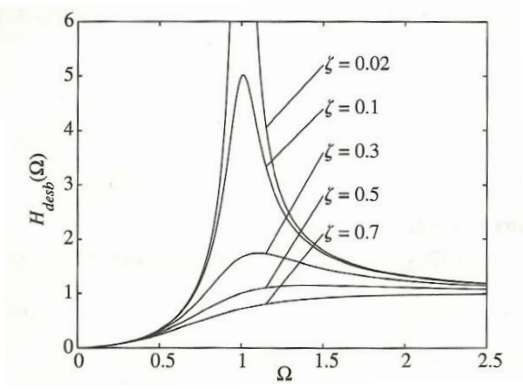
FIGURA NO. 22. DESPLAZAMIENTO EN RÉGIMEN PERMANENTE DE UN SISTEMA NO AMORTIGUADO

$$x(\tau) = M\epsilon * \frac{\Omega^2}{1-\Omega^2} * \text{sen}(\Omega\tau)$$

Cuando esto ocurre se corre un peligro muy grande y es que las vibraciones podrían llegar a ser destructivas si la frecuencia de excitación se encuentra cercana a la frecuencia de resonancia. Normalmente, cuando un sistema amortiguado está cerca de la resonancia la vibración se amplifica, pero en el caso de que no haya amortiguamiento el sistema podría llegar a ser caótico. En este caso Ω es igual a 1 y como se puede ver en la expresión anterior para el desplazamiento en régimen permanente la amplitud es infinita. Esto quiere decir, que el movimiento comienza a incrementar con el tiempo hasta un valor muy grande de oscilación. En la práctica esto significa que el material presentaría ruptura en algún momento y el sistema colapsaría. [5]

Una buena forma de determinar que tanto podría llegar a vibrar un sistema con desbalance es utilizando una gráfica como la de la Figura No. 23.

FIGURA NO. 23. RESPUESTA EN AMPLITUD DE UN SISTEMA VIBRATORIO A DIFERENTES FRECUENCIAS. [5]



Aún cuando no se conozca a la perfección la magnitud de la fuerza de desbalance, se puede intuir que si la función Hdesb(Ω) es pequeña la amplitud de la

vibración será menor que si la función Hdesb(Ω) es grande. Véase que si el amortiguamiento es mayor la amplitud decrece y si se encuentra cerca de la resonancia la amplitud aumenta.

4. FIBRA DE VIDRIO Y SU TRABAJO. La fibra de vidrio junto con la fibra de carbono y la fibra de aramida son las tres principales fibras que se utilizan como materiales de refuerzo. La fibra de vidrio es la de menor precio aunque no es la más liviana ni la más resistente, se utiliza para reforzar matrices plásticas y formar materiales compuestos para el moldeo. Las principales características que los materiales fabricados con fibra de vidrio presentan son las siguientes: buena relación resistencia/peso, buena estabilidad dimensional; buena resistencia al calor, al frío, a la humedad y a la corrosión; buenas propiedades aislantes eléctricas, facilidad de fabricación y costo relativamente bajo. Las fibras de vidrio comunes están compuestas por un 52-56% de SiO2, 12-16% de Al2O3, 16-25% de CaO y 8-13% de B2O3. Para obtener la fibra de vidrio, se debe estirar los monofilamentos que provienen de un horno que contiene vidrio fundido, luego se reúnen para formar un cordón. Las propiedades físicas de las fibras de vidrio se muestran en la Tabla No. 1 de manera comparativa con las fibras de carbono y las de aramida. [51]

TABLA NO. 1. PROPIEDADES FÍSICAS DE LOS MATERIALES [51]

Propiedad	Vidrio E (HTS)	Carbono (tipo HT)	Aramida (Kevlar 49)
Resistencia a la tensión, ksi (MPa)	350 (2410)	450 (3100)	525 (3617)
Módulo, Msi (GPa)	10 (69)	32 (220)	18 (124)
% de elongación a la rotura	3,5	1,40	2,5
Densidad, g/cm³	2,54	1,75	1,48

El trabajo en fibra de vidrio requiere de tres componentes principales para formar una estructura sólida, resistente y liviana. La primera es la resina de poliéster, esta por sí sola no produce mayor efecto, simplemente es un líquido viscoso. El segundo componente es el catalizador, que normalmente es un compuesto de peróxido, su función es reaccionar con la resina para que ésta se endurezca. Hay que tener especial cuidado con la aplicación del catalizador, si se utiliza poco catalizador la resina endurecerá, pero se obtendrá un material pegajoso, si se utiliza mucho catalizador el proceso de

endurecimiento se llevará a cabo mucho más rápido pero se obtendrá una mezcla no tan duro como la que se consigue si se siguen las especificaciones del fabricante, además el material podría llegar a presentar fractura con el tiempo. El tercer componente es la fibra de vidrio, este material se consigue en distintas presentaciones, como lazos, como un tejido o como pequeños trocitos. El procedimiento consiste en colocar la resina sobre un recipiente y añadir el catalizador en la proporción recomendada por el fabricante, luego colocar las mayas de fibra de vidrio sobre el molde base y untar la resina con una brocha para que se adhiera a la fibra, finalmente cuando ya esté toda la forma de la pieza a fabricar se debe dejar secar y se esperar a que endurezca para luego retirar el molde. [66]

FIGURA NO. 24. MALLA DE FIBRA DE VIDRIO.

FIGURA NO. 25. RESINA [52]

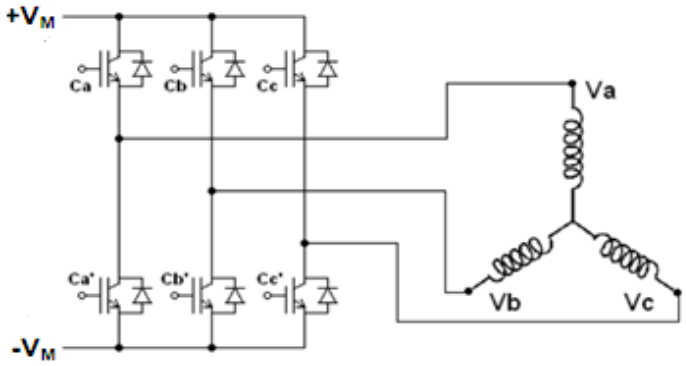
B. ESTABILIZACIÓN Y CONTROL

1. MOTORES DC SIN ESCOBILLAS. Los motores sin escobillas, son motores sincrónicos de tres fases, se componen por dos partes importantes, el estator y el rotor. El estator es la parte inmóvil del mecanismo y el rotor es la parte móvil. El estator está compuesto por varias ranuras en donde se enrollan grandes cantidades de cobre, a las que se les conoce como bobinas, cada uno de estos embobinados sirve como polos al momento de colocarse el rotor en su centro. De estos polos se obtienen tres terminales principales a las que se les colocan tres señales a 120 grados. El rotor es la parte giratoria en el motor, esta parte está compuesta por imanes permanentes iguales al número de bobinas en el estator. Normalmente los imanes son hechos con ferrita, por sus características magnéticas. [67]

El principio de funcionamiento de un motor DC sin escobillas, se basa en la atracción entre los diferentes polos. En el lado del estator se colocan tres señales desfasadas 120 grados. Cuando la corriente pasa sobre las bobinas, se crea un campo magnético que crea atracción y repulsión entre los imanes permanentes y las bobinas, cuya polaridad varía en el tiempo. Este movimiento se conoce como sincrónico ya que el rotor girará a la misma velocidad angular que la de las señales desfasadas. [67]

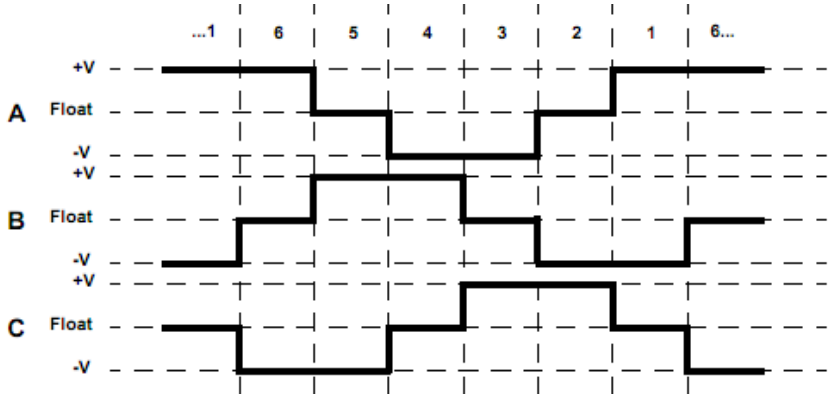
Usualmente las señales trifásicas en los motores son sinusoidales, sin embargo, para los motores DC sin escobillas, pueden ser sinusoides o no. Si se utilizan tres señales sinusoidales desfasadas 120 grados, el motor sería más bien un motor trifásico común y corriente. Si se utiliza una señal conmutada entonces se trata de un motor DC sin escobillas (BLDC). La idea detrás de este tipo de motores es utilizar un puente de tres fases tal y como se muestra en la Figura No. 26. [67]

FIGURA NO. 26. CONFIGURACIÓN TRIFÁSICA CON SEÑALES CONMUTADAS. [67]



Cada una de las señales que llega a los puntos Va, Vb y Vc puede ser VM+, VM- o ninguna conexión. De esta cuenta es posible generar una señal periódica de tres estados. Si a este resultado agregamos una combinación de conmutaciones en un orden particular tal que los tres estados estén desfasados para cada una de las señales, se obtienen tres señales cuadradas que simulan un desfase de 120 grados entre las tres terminales del motor, tal y como se observa en la Figura No. 27. [11]

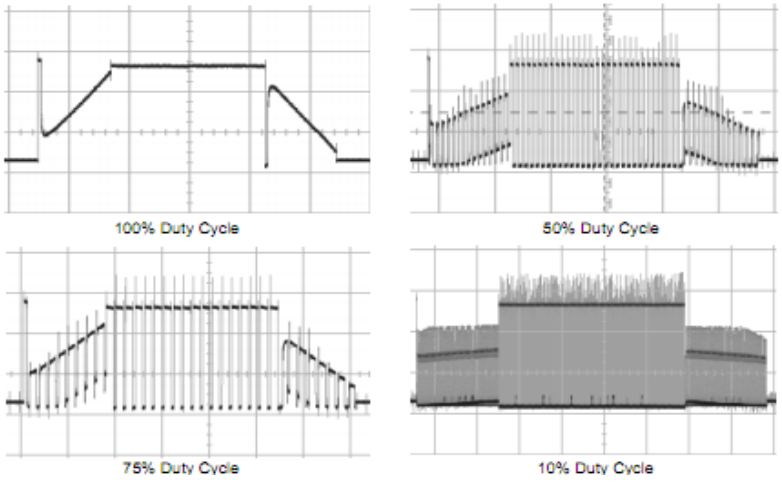
FIGURA NO. 27. SEÑALES TRIFÁSICAS GENERADAS CON PUENTE CONMUTADOR. [11]



Cuando este sistema se mantiene funcionando, el rotor gira a cierta velocidad angular que dependerá básicamente de la potencia que se entregue en la fuente de voltaje DC colocada en las terminales del puente trifásico. Por lo tanto un método común para obtener el control de velocidad es por medio de un PWM (ancho de pulso modulado), este tipo de señal permite modular la potencia que es entregada por la fuente DC del puente trifásico.

En la Figura No. 28 se observa como el principio de PWM es aplicado al control de un motor DC sin escobillas para modificar la potencia entregada al movimiento y por ende la forma del voltaje inducido en el estator.

FIGURA NO. 28. SEÑAL ENTRE DOS FASES DE MOTOR DC SIN ESCOBILLAS A DIFERENTES PWM. [11]



2. FUNCIONES DE TRANSFERENCIA PARA SISTEMAS DE CONTROL.

Cuando se desea tener un control preciso de alguna variable en un sistema, ya sea mecánico, químico, eléctrico, etc. Es necesario recurrir a la teoría de sistemas de control. Esta es una disciplina que busca modificar intencionalmente una variable para que permanezca en un valor de interés particular y que el sistema continúe estable a pesar de perturbaciones externas.

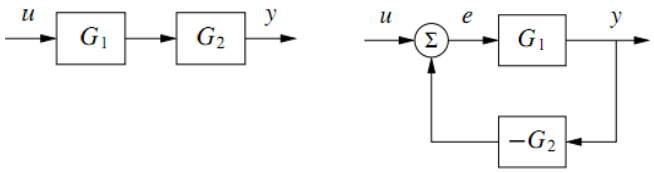
Dependiendo del método y tipo de control que se desee utilizar puede ser o no necesario un modelo matemático que describa al sistema o que lo aproxime. En general es deseable tener un modelo matemático que permita el análisis de la respuesta del sistema cuando es sometido a un impulso, tanto para sistemas con retroalimentación como para sistemas sin retroalimentación.

Cuando se trata de sistemas lineales e invariantes en el tiempo, la herramienta matemática que se utiliza es la transformada de Laplace. En principio esta herramienta es un método más de ecuaciones diferenciales, sin embargo, en sistemas de control forma parte de un método que permite generar funciones de transferencia. Una función de

transferencia no es más que la relación entre una señal de salida versus una señal de entrada en el dominio de Laplace. La primer ventaja que presenta este método es que transforma las ecuaciones diferenciales a polinomios que son sencillos de operar con algebra básica. Además permite modelar a los sistemas complejos por medio de pequeños sistemas interconectados en cascada. [37]

En general, los sistemas reales se pueden modelar por diversos bloques de transferencia, pueden ser simples como un amplificador o un derivador hasta filtros de varios polos. Por lo general, una función de transferencia posee información sobre el retardo del sistema, el ancho de banda de las -frecuencias que deja pasar y la amplificación o atenuación sobre las señales de entrada. Además una función de transferencia es una expresión matemática de dos polinomios "s" que se puede aplicar en general a todo tipo de sistema, esto quiere decir que si estamos hablando de un sistema mecánico, la función de transferencia podría ser la información sobre velocidades, aceleraciones, inercias, deflexiones, etc. En un sistema electrónico podría ser la información sobre voltaje, corriente, potencia, etc. o aún más podría ser la información sobre dos variables, una electrónica y una física. Por lo tanto, esta herramienta permite modelar un sistema completo para ser controlado posteriormente. En la Figura No. 29 se presenta un sistema sin retroalimentación y un sistema controlado por medio de retroalimentación. [37]

FIGURA NO. 29. MODELO DE UN CONTROL DE LAZO ABIERTO Y LAZO CERRADO. [37]



3. SISTEMAS DE CONTROL. Un sistema de control es un grupo de componentes que pueden regular su propia conducta o la de otro sistema o sistemas, con el fin de lograr un funcionamiento definido. Los principales objetivos de un sistema de control son: [68]

- Ser estables y robustos frente a perturbaciones y errores en los modelos.
- Ser eficiente según un criterio preestablecido evitando comportamientos bruscos.

Dos principales tipos de sistemas de control son los de lazo abierto y los de lazo cerrado. El sistema de lazo abierto depende solo de la entrada, mientras que el sistema de lazo cerrado depende también de su salida, por lo que tiene mayor estabilidad a perturbaciones. [68]

El sistema se suele visualizar como diagramas de bloques y se suelen analizar en el dominio de frecuencias, pero no se limitan a este análisis. [68]

El control de lazo cerrado más utilizado es el control PID. Este calcula el error entre el nivel deseado y el nivel de salida, y luego trata de minimizarlo. Este consiste de tres términos principales: [68]

- Proporcional: Crea un cambio de salida proporcional al error. Al aumentar su valor se puede obtener una respuesta más rápida y un error en estado estable menor. Si es demasiado grande entonces el sistema se puede volver inestable. [68]

FIGURA NO. 30. TÉRMINO PROPORCIONAL

$$P_o = K_p * e(t)$$

- Integral: Depende del error y de la duración del error. Este aumenta la velocidad hacia el nivel deseado y elimina el error en estado estable. Puede crear sobreelevaciones. Suele crear un efecto de sobre elevación cuando el error varia de signo, por lo que se recomienda utilizar un modelo con ventanas de valores máximos y mínimos (saturaciones). [68]

FIGURA NO. 31. TÉRMINO INTEGRAL

$$I_o = K_i * \int_0^t e(\tau) d\tau$$

- Derivativo: Depende del ritmo de cambio del error. Reduce las sobreelevaciones del sistema y mejora la estabilidad del control. Puede

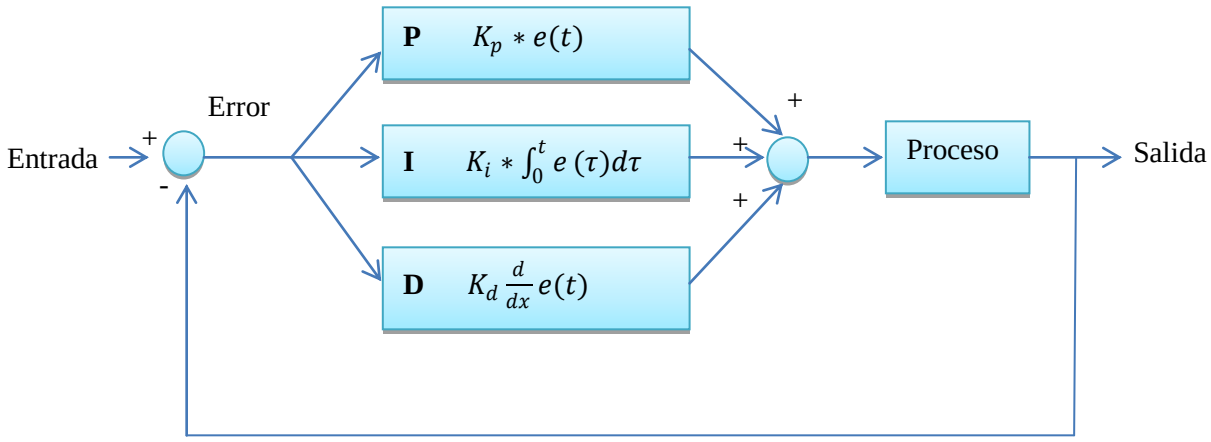
reducir el tiempo de la respuesta y amplifica señales de ruido. Para evitar el ruido se suele utilizar un modelo de derivada con ancho de banda limitado. [68]

FIGURA NO. 32. TÉRMINO DERIVATIVO

$$D_o = K_d * \frac{d}{dt} e(t)$$

El control PID es usualmente la mejor opción cuando no se tiene conocimiento sobre el sistema a controlar. El método más utilizado en estos casos es el método de sintonización de Ziegler-Nichols y sus variantes. Este requiere que se aumente la constante proporcional hasta que el sistema oscile y luego propone valores iniciales para las demás constantes. Otro método similar genera un modelo del sistema midiendo la variable de salida en función de la de entrada, analizando la amplitud, retardo y amortiguamiento del sistema, analizando sus polos dominantes. En la Figura No. 33 se presenta un modelo de un control PID clásico. [68]

FIGURA NO. 33. MODELO DE UN CONTROL PID



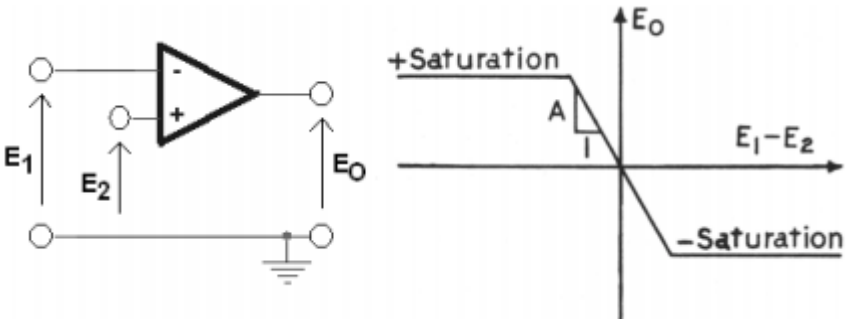
4. AMPLIFICADORES OPERACIONALES Y SUS CONFIGURACIONES. Un amplificador operacional es básicamente un filtro, esto quiere decir que posee características de amplificar, desfasar y dejar pasar ciertas frecuencias que se encuentren dentro del rango de su ancho de banda. Su modelo real se puede escribir como sigue.

FIGURA NO. 34. MODELO MATEMÁTICO DEL AMPLIFICADOR OPERACIONAL

$$\frac{Vo}{Va} = \frac{Ao}{1 + j\frac{\omega}{\omega a}}$$

Donde ωa, es la frecuencia angular máxima que el filtro deja pasar sin que su atenuación sea mayor a 3 decibeles, Ao es la ganancia del amplificador operacional, Va es el voltaje aplicado entre sus terminales y Vo es el voltaje de salida. Cuando la frecuencia es muy baja el amplificador operacional prácticamente solo amplifica la señal. Dado que este dispositivo electrónico está limitado por la alimentación que se le proporcione, la amplificación no se da para todos los valores de Va, solamente funciona cuando el voltaje aplicado se encuentra en la región proporcional del amplificador, de lo contrario el amplificador se satura a un voltaje positivo o negativo. En la Figura No. 35 se observa la curva característica de un amplificador operacional real. [10]

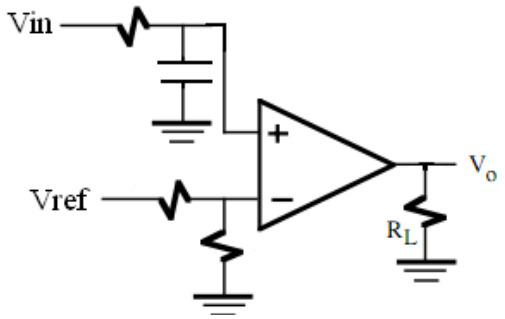
FIGURA NO. 35. CURVA CARACTERÍSTICA DE UN AMPLIFICADOR OPERACIONAL. [10]



Dado que la ganancia del amplificador es grande, solamente existe una región de proporcionalidad pequeña antes de saturarse. Si se coloca un voltaje en configuración sin retroalimentación, podría ser complejo establecer un voltaje muy pequeño para que el amplificador opere en la región proporcional, sin embargo, al colocar una retroalimentación negativa es posible hacer que el amplificador operacional opere en esta región y se consiga realizar una gama de funciones con este dispositivo. Un amplificador operacional tiene algunas características importantes: Su impedancia de entrada es muy grande lo que permite realizar circuitos aislados, es decir, que si se colocan en cascada uno no cargue al otro. Por otra parte la corriente necesaria para hacer funcionar a un circuito siguiente será proporcionada por una fuente de corriente interna en la salida del amplificador operacional. [10]

Algunas de las funciones que se pueden lograr con el amplificador operacional son: sumar, restar, multiplicar, integrar, derivar, filtrar, comparar, etc. A continuación se presenta una configuración del amplificador operacional útil para comparación con señales filtradas.

FIGURA NO. 36. AMPLIFICADOR OPERACIONAL EN CONFIGURACIÓN COMPARADOR. [10]



En este circuito, el voltaje en la terminal negativa del amplificador operacional es ajustable según se desee y la señal de entrada en la terminal positiva es filtrada a una frecuencia de corte calculada con la siguiente ecuación.

FIGURA NO. 37. FRECUENCIA DE CORTE

$$\omega c = \frac{1}{RC}$$

5. FILTROS DIGITALES. Un filtro digital es un sistema que aplica operaciones matemáticas a una señal muestreada y discreta para atenuar o amplificar ciertos aspectos de ella. Se caracterizan por una función de transferencia o su ecuación de diferencias.

Los dos tipos principales de filtros son los FIR (respuesta finita al impulso) y los IIR (respuesta infinita al impulso). Los FIR tienen la ventaja que no tienen retroalimentación, son inherentemente estables y son fácilmente diseñados para obtener un ángulo de desfase lineal. Los filtros IIR si utilizan retroalimentación, y necesitan un menor número de polos para obtener la misma respuesta que un filtro FIR. [50]

Debido a la retroalimentación, es necesario estudiar la estabilidad de los filtro IIR. Un requisito para la estabilidad es que los polos se encuentren adentro del círculo

unitario, en otras palabras, sus polos deben ser de magnitud menor a uno. La función de transferencia para un sistema causal, lineal e invariante en el tiempo es expresada en el dominio Z de la siguiente manera: [50]

FIGURA NO. 38. FUNCIÓN DE TRANSFERENCIA EN DOMINIO Z

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_M z^{-M}}$$

Luego hay que comprobar que todos sus polos se encuentren adentro del círculo unitario. A continuación se detalla la determinación de la frecuencia de corte de cada polo:

FIGURA NO. 39. DETERMINACIÓN FRECUENCIA DE CORTE

$$z^n = e^{nTs}$$

En el caso de un filtro pasa baja de primer orden:

$$\frac{1}{s + a} = \frac{1}{1 - e^{-aT} z^{-1}}$$

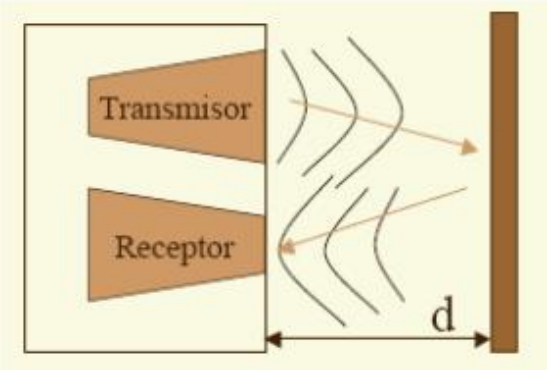
Dónde

- a es la frecuencia de corte
- T el período de muestreo

6. SENSORES ULTRASÓNICOS. Un sensor ultrasónico es aquel que permite medir la distancia por medio de los ultrasonidos. El sonido que el ser humano puede escuchar se encuentra en un rango de frecuencias que van desde 16 Hz hasta 20KHz, sin embargo, para los ultrasonidos se utilizan frecuencias mayores a las audibles, por ejemplo, frecuencias de 40kHz. Un sensor ultrasónico es básicamente un acoplamiento de bocina y micrófono, al que se le conoce como transmisor y receptor. El procedimiento para obtener la distancia se basa en conocer el tiempo de emisión de una onda para luego medir el tiempo en el que el receptor escucha su eco. De esta forma conociendo la velocidad del sonido en el aire y el tiempo es posible determinar la distancia que ha recorrido por medio de la siguiente ecuación. [41]

FIGURA NO. 40. ESQUEMA DE FUNCIONAMIENTO DEL SENSOR ULTRASÓNICO. [41]

$$d = \frac{Vt}{2}$$



Los sensores ultrasónicos no son útiles cuando se requiere precisión en una aplicación, sin embargo, cuando solamente se requiere detección de objetos o cuando las condiciones ambientales no afectan demasiado la medición, estos sensores son muy útiles por su sencillez. Los sensores ultrasónicos al igual que las bocinas de audio funcionan dentro de un cono de emisión, esto significa que la distancia que el sensor podría percibir no es exactamente la que se encuentre frente a él sino que podría ser la distancia del objeto más próximo que detecte dentro cono acústico. Además, la respuesta varía del material sobre el cual la onda rebote y también del ángulo con que esta onda rebote ya que la onda podría no regresar directamente hacia el sensor. [39]

7. CONTROLADOR DEL MOTOR. Utilizando el controlador "*TURNIGY Plush 25amp Speed Controller*" (Figura No. 41), se logró realizar el control del motor. Este controlador permite una fácil interacción con el motor sin escobillas, utilizando PWM (modulación de ancho de pulso por sus siglas en inglés).

FIGURA NO. 41. CONTROLADOR DE VELOCIDAD

El controlador tiene la característica que soporta una descarga continua de 25A para controlar los motores utilizados. Uno de sus beneficios es que este controlador proporciona una salida de 5V regulada a 2A, por lo que pueden ser utilizados para alimentar circuitos externos. [61]

Otro de sus beneficios es que este driver regula la potencia que consume el motor, controlando la fuerza de sustentación otorgada por las hélices. Esto crea una función de transferencia lineal hacia fuerza, simplificando el análisis y control del cuadcóptero.

8. MICROCONTROLADOR UTILIZADO. Se optó por usar un dsPIC30f4011 para controlar el cuadcóptero. Este es un microcontrolador de 40 pines, de la familia de los dsPIC30F, especializadas en procesamiento de señales digitales. Utilizan una arquitectura de 16 bits y tienen módulos especializados para el procesamiento de las señales. Este presento varias ventajas respecto a otros tipos de microcontroladores. Entre estas se puede mencionar: [31]

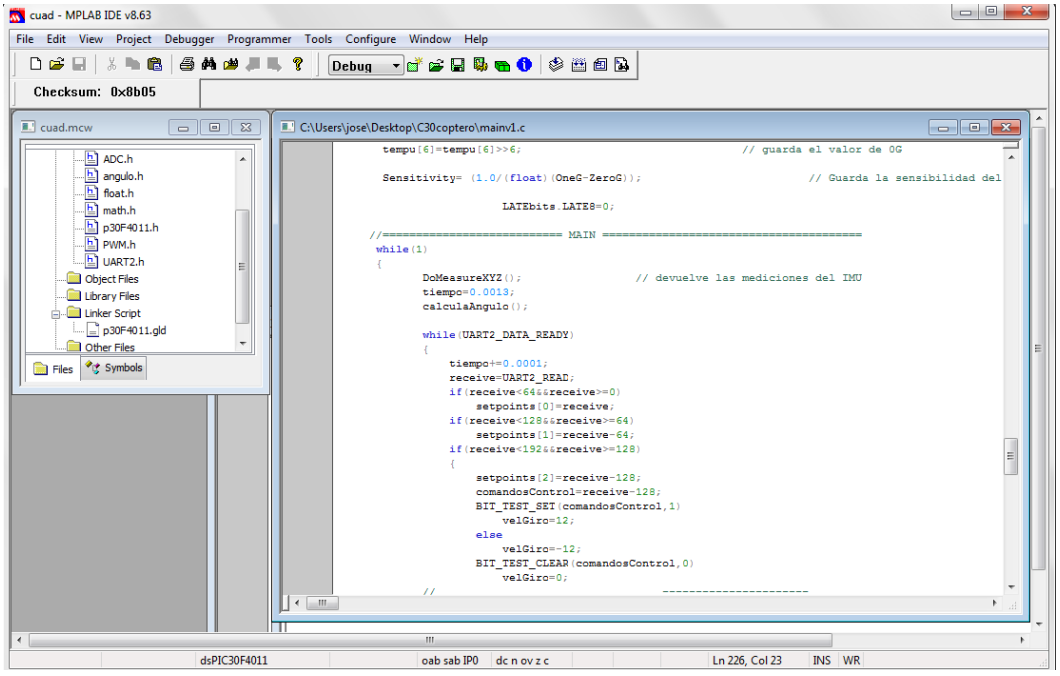
- Multiplicación y división implementada en hardware para una mayor velocidad de procesamiento.
- Tres salidas de PWM de control de motor (PWM-MC) y una salida de PWM del módulo CCP. Juntos se obtenían las 4 salidas de PWM requeridas para controlar todos los motores del cuadcóptero.
- Nueve canales de ADC (conversión analógica digital) con resolución de 10 bits y una velocidad de conversión de 500ksps, aumentable a 1msps con bajas impedancias.

- Una velocidad de hasta 30 MIPS, utilizando reloj externo y un PLL de x16. Se utilizó un reloj de 80 MHz, llegando a 20 MIPS.
- Arquitectura del Set de Instrucciones optimizada para el compilador C
- Modos de direccionamiento flexibles.

9. LENGUAJE DE PROGRAMACIÓN EN EL MICROCONTROLADOR. Se utilizó el IDE de MPLAB, con lenguaje C30. Una imagen del IDE se muestra en la Figura No. 42. Se eligió este lenguaje basando en "C" debido a:

- Mayor facilidad de programación que el lenguaje ensamblador
- Complejidad del código
- ISA optimizada para el compilador en C
- Alto control del microcontrolador.
- Recomendado para la programación en esta familia de microcontroladores.
- Facilidad de organización del código, utilizando archivos auxiliares "header files".

FIGURA NO. 42. MPLAB IDE CON COMPILADOR C30



10. SENSORES DE ESTABILIZACIÓN. La estabilización del cuadcóptero se llevó a cabo por medio de la obtención del ángulo de inclinación del mismo. Para lograr obtenerlo fue necesario utilizar un acelerómetro y un giroscopio. Por medio de cada uno se obtuvo una aproximación del ángulo y se obtuvo una aproximación más fiable al tomar en cuenta ambas mediciones. El módulo IMU (unidad de medición inercial) de 6DOF utilizada se muestra en la Figura No. 43.

FIGURA NO. 43. IMU 6DOF [40]

a. Acelerómetro. Un acelerómetro es un aparato que mide la aceleración propia de un objeto, es decir, la aceleración física experimentada por el mismo. No es necesariamente la misma aceleración de un eje de coordenadas, si no que el peso experimentado por una masa de prueba en el plano de referencia del acelerómetro. Es decir, El acelerómetro mide el peso por unidad de masa, también conocida como fuerza específica.

Debido a esto es que un acelerómetro en estado estático mide una aceleración de 1g hacia abajo, y en caída libre no presenta aceleraciones. Esto es porque en caída libre, aunque su velocidad este aumentando, también aumenta la del objeto en su interior. Por lo que continua midiendo respecto a su mismo plano de referencia, donde el objeto pareciera mantenerse estático.

Se puede obtener su orientación utilizando un acelerómetro de tres ejes, utilizando su propiedad de medir una aceleración de 1g al estar estático. Para obtenerla se puede utilizar trigonometría.

El acelerómetro utilizado es un ADXL335. Entre sus principales características se encuentran: [40]

- Medición de 3 ejes
- Salidas analógicas
- Tiene un rango de ±3 g
- Sensibilidad de 300mV/g
- Frecuencia de corte de 550Hz

b. Giroscopio. Un giroscopio es un dispositivo utilizado para medir o mantener una orientación. Los giroscopios más sencillos se basan en los principios de conservación de momento angular, pero hoy en día los más comunes y económicos son los de tecnología MEMS. Los giroscopios se usan en aplicaciones donde los compases magnéticos podrían no funcionar, para estabilización de vehículos aéreos, mantener la dirección en la minería, y otros. Los giroscopios nos permiten obtener una lectura de la velocidad angular. Esta puede ser integrada para obtener una aproximación del ángulo. [64]

Los giroscopios utilizados fueron el LPR530AL y el LY530AL. El primero es un giroscopio de dos ejes y el segundo es un giroscopio sobre el eje restante. Tienen las siguientes características: [68]

- Medición conjuntamente en los 3 ejes
- Salida analógica
- Tiene un rango de ±300°/s
- Sensibilidad de 3.33 mV/°/s
- Frecuencia de corte de 140Hz

11. OBTENCIÓN DEL ÁNGULO DE INCLINACIÓN. Los datos obtenidos por el acelerómetro y el giroscopio tienen desventajas que no permiten utilizarlos individualmente. El acelerómetro tiene una medición ruidosa del ángulo

debido a otras aceleraciones externas, que no suelen ser constantes. El giroscopio tiene un problema de sesgo al momento de integrarlo, lo que lo hace buen sensor en estado dinámico pero no confiable estáticamente. [64] [25]

Ambos problemas se resuelven idealmente por medio de un filtro Kalman. Este es un método para obtener una medición indirecta que podría tener ruido. Es un algoritmo multivariable que hace usos de matrices de covarianzas para obtener estimados del siguiente estado, y requiere un modelo del sistema para obtenerlo. [65]

Este método se vuelve complicado de entender y aplicar eficientemente, por lo que es necesario un enfoque más intuitivo. Se utilizó un filtro complementario o filtro ponderado. Este filtro se puede reducir a una forma bastante simple y su implementación es sencilla. Este filtro puede ser descrito con la siguiente ecuación: [25] [14]

FIGURA NO. 44. MODELO DE FILTRO PONDERADO

$$Y = a * X_0 + (1 - a)X_1$$

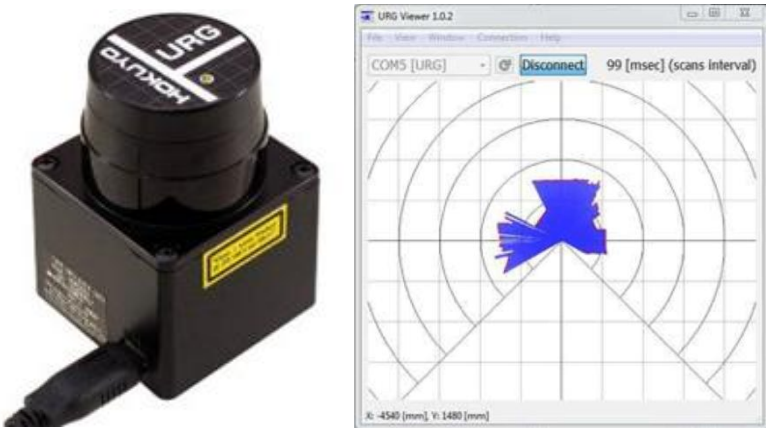
Dónde

- X_0 es un estimado utilizando un primer instrumento de medición
- X_1 es un estimado utilizando un segundo instrumento
- a es el nivel de confianza en cada sensor

C. SENSOR E INTERFAZ RS-232

1. SENSOR LÁSER DE DISTANCIA. El sensor de distancia usado es un Hokuyo URG-04LX-UG01 (Figura No. 45). Este sensor funciona como un radar láser con un área de detección de 240° y su rango de medición está comprendido desde 20 mm hasta 5.6 metros. La fuente de luz del sensor es un láser infrarrojo que permite medir distancias a una frecuencia alrededor de los 4 kHz.

FIGURA NO. 45. HOKUYO URG-04LX-UG01



2. FORMATO DE COMUNICACIÓN DEL SENSOR. El sensor láser implementa 15 comandos de comunicación agrupados según su función en el protocolo SCIP 2.0. Este protocolo especifica la forma en que se codifican y decodifican los datos de distancia y el formato de comunicación entre el sensor y su controlador.

FIGURA NO. 46. FORMATO DE COMUNICACIÓN DESDE EL CONTROLADOR HACIA EL SENSOR [23]

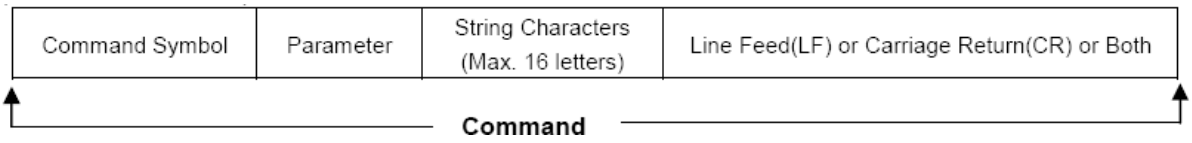
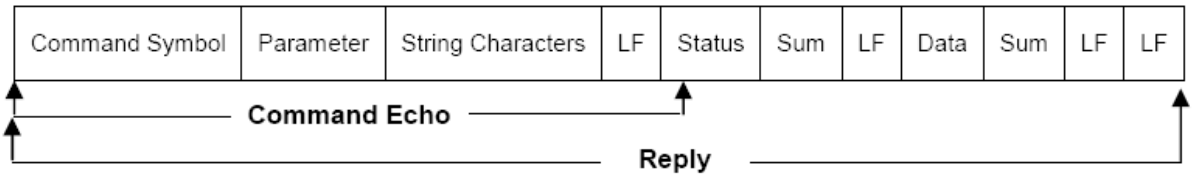


FIGURA NO. 47. FORMATO DE COMUNICACIÓN DESDE EL SENSOR HACIA EL CONTROLADOR [23]



Las características principales del formato de comunicación son las siguientes:

TABLA NO. 2. CARACTERÍSTICAS PRINCIPALES DEL FORMATO DE COMUNICACIÓN DEL SENSOR LÁSER

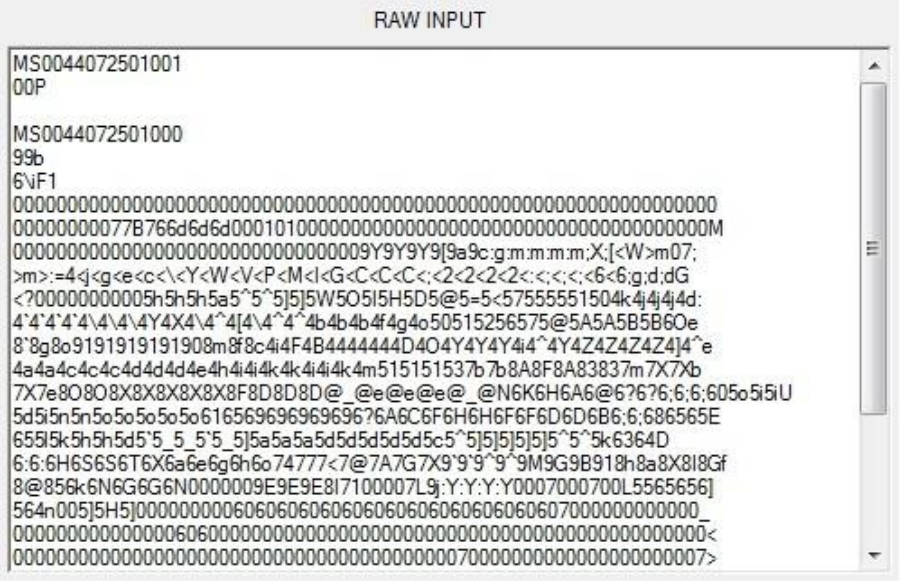
Característica	Longitud (Bytes)	Descripción
Command Symbol	2	Es un código al inicio de cada comando. Cada comando tiene su propio código de identificación (opcode).
Parameter	1-13	Es información requerida para modificar la configuración del sensor (operandos).
Line Feed (LF)/ Carriage Return (CR)	1 o 2	Es un código de terminación, puede contener ambos caracteres. La respuesta siempre tendrá dos LF continuos.
Status	2	Indican el estado del procesamiento de la última instrucción recibida desde el host. Informan acerca de errores de autenticación, recepción de comando inválida o número de operandos desconocido.
Sum	1	Se utiliza para la autenticación de los datos.
Data	≤64	Es la información relacionada al último comando enviado. Está separada por un LF y SUM después de cada 64 bytes de datos.

Acerca de la característica SUM:

El byte de SUM se calcula sumando todos los datos recibidos entre dos caracteres LF, luego es necesario convertir dicha suma a un número binario y tomar los 6 bits menos significativos de dicho número. Finalmente, sumamos 30H al valor obtenido en el paso anterior.

La Figura No. 48 muestra el formato de los datos recibidos desde el sensor laser.

FIGURA NO. 48. INFORMACIÓN RECIBIDA DESDE EL SENSOR LÁSER



D. MAPEO Y EXPLORACIÓN

1. PERCEPCIÓN ROBÓTICA. La percepción es el proceso con el cual robots o sistemas mapean mediciones de sensores en representaciones internas del ambiente. La percepción tiene un alto grado de dificultad debido a que en general, los sensores son ruidosos, y el ambiente es parcialmente inobservable, impredecible y muchas veces dinámico. [44]

Una buena representación del ambiente tiene tres propiedades: [44]

- Contiene suficiente información para que el robot pueda tomar decisiones acertadas.
- Está estructurada de manera que pueda ser actualizado eficientemente.
- Es natural, en el sentido de que variables internas corresponden a variables de estado natural en el mundo físico.

a. Localización. La localización es un caso genérico de la percepción robótica. Es el problema de determinar en donde están las cosas. El problema de la localización es uno de los más trabajados en el área de la percepción robótica,

debido a que el conocimiento de la ubicación de las cosas es el núcleo del éxito de cualquier interacción física. Por ejemplo, los robots de navegación deben saber en donde se encuentran para poder encontrar su camino hacia posiciones objetivo. [44]

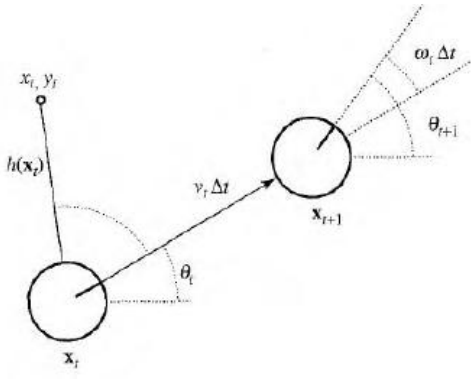
El problema de localización, se presenta en tres niveles de dificultad, y tienen una denominación distinta de acuerdo a sus características: [44]

- **Problema de rastreo**: Este caso se da cuando se tiene conocimiento de la posición inicial del objeto a localizar. En este problema la incertidumbre está limitada.
- **Problema de localización global**: No se tiene conocimiento de la posición inicial del objeto. Involucra etapas con incertidumbre muy amplia, pero una vez se logra localizar al objeto, se vuelve un problema de rastreo.
- **Problema del secuestro**: Este es el caso de mayor dificultad, en el cual el objeto a localizar es "secuestrado" o retirado del ambiente. Generalmente se aplica este problema para evaluar la robustez de técnicas de localización.

Una característica general de los problemas de localización, es que el robot cuenta con un mapa exacto del ambiente.

b. Modelo planteado – Enfoque a localización. Aplicando el problema de localización al caso específico de un robot móvil que se está trasladando en un ambiente dado, en dos dimensiones, se puede plantear un modelo para el sistema.

FIGURA NO. 49. MODELO CINEMÁTICO DE UN ROBOT MÓVIL. [44]



La posición planteada por el modelo está definida por sus coordenadas cartesianas con valores x y y y su dirección con valor θ, como se muestra en la Figura No. 49. Este modelo está simplificado, pues en realidad debería de ser un modelo dinámico, pues se está moviendo. Las acciones en este modelo, consisten en la especificación instantánea de dos velocidades, una velocidad de traslaciónv, y una velocidad rotacional w, en intervalos de tiempo pequeños. Por lo tanto, un modelo aproximado para el movimiento del robot, está dado por: [44]

FIGURA NO. 50. MODELO PARA MOVIMIENTO DEL ROBOT. [44]

$$\boldsymbol{X}_{t+1} = f(\boldsymbol{X}_t, v_t, w_t) = \boldsymbol{X}_t + \begin{pmatrix} v_t \Delta t \, cos\theta_t \\ v_t \Delta t \, sin\theta_t \\ w_t \Delta t \end{pmatrix}$$

El modelo proporciona la siguiente posición del robot, dada en función de su posición actual, y una acción representada por una velocidad lineal y una angular. La notación $\boldsymbol{X}_t$ se refiere a un estado del sistema.

Para el problema, también es necesario plantear un modelo para el sensor, la entrada de información al robot. Este modelo puede variar mucho dependiendo del tipo de sensor utilizado, pero especificando a un sensor láser el modelo se plantea de la siguiente manera:

El sensor proporciona un vector de datos de rango (distancia) $\boldsymbol{z}_t = (z_1, \ldots, z_n)$, con un ángulo asociado a cada posición en el vector, relativo a la posición del robot. El dato del ángulo no se incluye en el vector, éste simplemente se relaciona con el índice de

cada dato de distancia. Estos datos, están distorsionados por ruido Gaussiano inherente a cualquier sensor. [44]

2. MAPEO – PROBLEMA SLAM. En el caso en que un robot no cuenta con un mapa de su ambiente, el problema de localización se convierte en un problema de mapeo. Ahora el robot debe construir el mapa del ambiente por sus propios medios además de localizarse en el mismo. Este problema es conocido en la literatura como el problema de localización y mapeo simultáneo, SLAM por sus siglas en inglés. [44]

El problema de SLAM se planteó en la década de los 90's, y es definido por Newman (2006:1) como sigue:

> «Se pregunta si es posible para un vehículo autónomo empezar en una localización desconocida en un ambiente desconocido, e incrementalmente construir un mapa de este ambiente mientras se usa el mismo mapa para computar la posición absoluta del vehículo. » [38]

El problema de SLAM es uno de los problemas principales en la robótica, y a pesar de que se han planteado muchas soluciones al mismo, los investigadores siguen trabajando en él para mejorar sus resultados. El motivo del alto interés en el mismo, es que su solución óptima va a permitir un avance significativo en el área.

El caso más sencillo del problema, que por su naturaleza ya es bastante difícil, se da cuando se pretende mapear un ambiente invariante. El problema se vuelve muy complejo cuando el ambiente varía mientras el robot se mueve. En el caso estático, el mapeo se vuelve un problema de inferencia Bayesiana, al igual que el problema de localización. Es decir que la posición o estado del robot y el mapa del ambiente, van a estar dados en función de los datos obtenidos por el sensor, y la posición o estado anterior. [44]

Una de las dificultades de este problema, es que el robot no tiene conocimiento de las dimensiones del ambiente que va a mapear, por lo que se debe ajustar dinámicamente sus dimensiones a lo largo del proceso de mapeo. [44]

3. PLANEACIÓN DE MOVIMIENTO IDEAL. En la robótica, la toma de decisiones lleva al movimiento o accionamiento de actuadores. El movimiento puede ser de dos tipos: [44]

- Movimiento punto a punto: Consiste en llevar el robot o alguno de sus actuadores a una posición meta. Es la traslación o cambio de configuración de un estado a otro.
- Movimiento por compatibilidad (**"***compliant motion***"**): Consiste en un cambio de configuración o posición mientras el robot está en contacto con algún obstáculo.

El problema de la planeación de movimientos o caminos, es encontrar un camino de una configuración o estado a otro en el espacio de configuraciones (determinado por localización, orientación y ángulos de uniones). En la robótica, una de las características principales del problema es que involucra espacios continuos. [44]

Existe un amplio rango de técnicas enfocadas en encontrar caminos en espacios continuos. Las principales familias de éstas técnicas se conocen como descomposición por celdas, y esqueletización. Cada familia reduce el problema de planificación de caminos a un problema de búsqueda en árboles discretos por medio de la identificación de estados canónicos y caminos en el espacio de configuración libre. [44]

4. ESPACIO DE CONFIGURACIÓN. Resulta indispensable el planteamiento de una representación apropiada del problema de planeación de movimiento para poder llegar a una solución. El conjunto de variables que caracterizan el estado de un robot en coordenadas cartesianas se conoce como la representación del espacio de trabajo. Esta representación resulta útil en aplicaciones como prevención de colisiones, sin embargo se tiene el problema de que no todas las coordenadas en el espacio de trabajo son alcanzables, ya sea por obstáculos o por las restricciones físicas del robot. [44]

Si se trata de planificar movimientos sobre este espacio de trabajo, se corre el riesgo de nunca llegar al objetivo, pues el camino para llegar a la solución puede pasar por una coordenada inalcanzable. Esto hace necesario plantear una mejor representación para el problema, y es aquí en donde surge la representación del espacio de configuración. [44]

El espacio de configuración describe el estado de un robot por la configuración de las juntas o uniones del robot. De esta manera, el espacio permite planificar movimientos teniendo la seguridad de llegar a cualquier estado válido deseado, siempre y cuando no haya obstáculos en el camino. Sin embargo, los espacios de configuración conllevan ciertas dificultades. Generalmente cuando se busca mover un robot se establece su movimiento en términos del espacio de trabajo, es decir en coordenadas cartesianas, y en ciertas ocasiones resulta un reto realizar la transformación de estas coordenadas al espacio de configuración. Esa transformación se conoce como cinemática inversa. Otro problema, resulta en la representación de los obstáculos del ambiente en el espacio de configuración. Estos pueden ser objetos muy simples en el espacio de trabajo, pero se vuelven en formas muy complejas en el espacio de configuración.

FIGURA NO. 51. ROBOT EJEMPLO, CON DOS GRADOS DE LIBERTAD. [44]

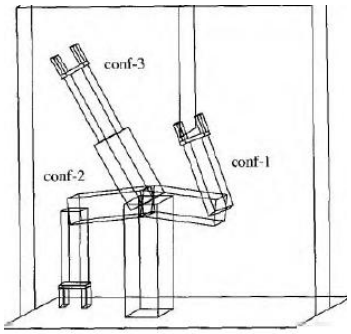
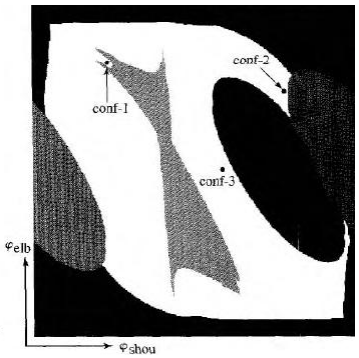


FIGURA NO. 52. ESPACIO DE CONFIGURACIÓN DEL ROBOT. [44]



Un espacio de configuración tiene dos regiones a diferenciar principales. La región o espacio libre, representada en blanco, y la región en grises y negro llamada espacio ocupado. El espacio libre corresponde al conjunto de todas las configuraciones que un robot puede asumir, y el ocupado a los obstáculos y configuraciones inaccesibles. [44]

La solución para trabajar con espacios de configuración, resulta entonces en no construir el espacio de configuración, sino simplemente examinarlo. Un algoritmo planificador entonces va a generar una configuración en el espacio de configuración, posteriormente transformarla al espacio de trabajo por medio de cinemática, y revisar el resultado final para eliminar estados prohibidos o inexistentes. [44]

5. PLANEACIÓN, MOVIMIENTO CON INCERTIDUMBRE.

Los algoritmos de planeación de movimiento mencionados hasta el momento no toman en cuenta una característica de los robots reales que cambia de manera significativa el análisis, la incertidumbre. La incertidumbre en la robótica surge de la observación parcial del ambiente y de efectos estocásticos de las acciones del robot. Otra fuente de incertidumbre son los errores que surgen de algoritmos de aproximación, que no le dan al robot un conocimiento exacto de su estado. [44]

Debido a la complejidad de los análisis probabilísticos al añadir la incertidumbre a sus ecuaciones, la toma de decisiones en la robótica usa algoritmos determinísticos como los mencionados previamente. Para poder hacer esto, es muy común que se realice una extracción del estado más probable de la distribución de estados producida por algoritmos de localización. Esto presenta ventajas computacionales que permiten resolver de manera eficiente los problemas de planeación simples. Una condición necesaria para poder ignorar la incertidumbre de esta manera, es que sea muy pequeña. [44]

En el caso en que la incertidumbre sea significativa, como el problema SLAM en donde el robot no tiene información de su posición, se debe de proponer una solución más robusta. No se puede ignorar la incertidumbre pues esto puede ocasionar efectos no

deseados, como colisiones con obstáculos. Se han desarrollado diversas técnicas para resolver estas situaciones, pero cada una presenta condiciones distintas para asegurar su correcto funcionamiento.

6. BÚSQUEDA EN GRAFOS APLICADA A PROBLEMA DE EXPLORACIÓN.

El espacio de configuración de un robot, puede ser simplificado a estructuras de datos sencillas como los grafos. Al lograr abstraer el espacio a este nivel, es posible aplicar la teoría de grafos y algoritmos de búsqueda para resolver el problema de exploración.

Un grafo, es un conjunto de vértices en el espacio, conectados por líneas llamadas aristas. Estos vértices pueden representar cualquier cosa, pero para el problema especificado, representan un estado de configuración de un robot. Las aristas representan movimientos que cambian al robot de un estado a otro.

Los algoritmos de búsqueda en grafos se pueden clasificar en dos ramas principales, los informados y los no informados. Los algoritmos no informados realizan una búsqueda sistemática pero sin información, mientras que los informados o heurísticos, cuentan con información que permite enfocar el proceso deductivo hacia una solución rápida, dirigiendo la búsqueda y evitando revisar vértices poco prometedores. [45]

Una característica general para los algoritmos de búsqueda en grafos, es que se debe de tener un vértice meta, es decir se debe de tener un objetivo de búsqueda.

a. Algoritmos de búsqueda no informada.

El término no informado, indica que no se cuenta con información adicional de los estados del grafo más que la dada por la definición del problema. Lo que realizan es generar sucesores a vértices y distinguir un estado meta de estados no meta. La diferencia entre las estrategias es el orden en que se expanden los vértices o nodos. [45]

Existen cinco algoritmos de búsqueda no informada que se consideran los más comunes y más utilizados, estos son:

- *Búsqueda primero en amplitud*: Esta estrategia se expande el nodo inicial, y luego todos los sucesores de éste se expanden, y luego sus sucesores, y así sucesivamente. En general todos los nodos se expanden a una profundidad dada antes de expandir los nodos en el siguiente nivel. [45]
- *Búsqueda de costo uniforme*: Cuando el grafo tiene una función de costo, el algoritmo de costo uniforme busca expandir siempre el nodo con el costo de camino más bajo. Si todos los costos son iguales, el algoritmo es idéntico al de búsqueda primero en amplitud. [45]
- *Búsqueda primero en profundidad*: Siempre expande el nodo más profundo en la rama actual. Al llegar a un final, regresa a un nivel anterior hasta haber explorado todos los nodos. [45]
- *Búsqueda de profundidad limitada*: Cuando se tiene un árbol o grafo sin límites, se puede establecer una profundidad límite para hacer posible la aplicación del método. Los nodos en el límite se considera como que no tuvieran sucesores, y se sigue analizando el resto del árbol en lugar de expandir a una profundidad mayor. [45]
- *Búsqueda de profundización iterativa*: Esta técnica llama a la búsqueda de profundidad limitada, y si no encuentra el objetivo, aumenta el límite y sigue con la búsqueda y esto se repite hasta encontrar la meta. [45]

b. Algoritmos de búsqueda informada. Los algoritmos de búsqueda informados, cuentan con una heurística o información que le permite al algoritmo de búsqueda tomar decisiones más eficientes para llegar a una meta. La heurística es una función que le asigna a cada nodo un valor correspondiente a algún parámetro relacionado con el problema de manera específica. [45]

Los algoritmos más comunes son:

- *Búsqueda primero el mejor*: Este algoritmo se asemeja a la búsqueda no informada en profundidad, pues va a expandir un solo nodo hasta llegar a un tope, y en este momento regresa a nodos anteriores para continuar con la búsqueda. La diferencia es que el nodo a expandir no va a ser el primero que encuentre, sino el que tenga la heurística más baja. [45]
- *Búsqueda A**: Este algoritmo es una variación de la búsqueda primero el mejor, en el sentido en que la heurística por la cual elige el nodo a expandir, es la suma del costo de moverse al nodo, con el costo del nodo. De esta manera búsqueda la heurística combinada más baja para elegir. [45]
- *Búsqueda escalando montaña*: Este algoritmo busca siempre mejorar (aumentar) el valor de la heurística actual, por lo que expande siempre el nodo con mayor heurística hasta llegar a un pico. Este algoritmo puede quedarse estancado en distintas situaciones, tales como máximos locales, o varios nodos con la misma heurística. [45]
- *Búsqueda en haz*: Este algoritmo expande todos sus sucesores en cada estado, y selecciona los *k* mejores nodos en función de su heurística. Se detiene hasta encontrar el estado meta. [45]

7. LENGUAJE DE PROGRAMACIÓN C#. C# de Microsoft, es un lenguaje de programación basado en el sistema o marco .NET. Es una evolución de los lenguajes C y C++ que presenta ventajas de simplicidad y seguridad. Es un lenguaje de programación orientado a objetos. [32]

Otra de sus ventajas, es la facilidad que presenta para comunicarse con distintos tipos de puertos y protocolos. En específico, la comunicación con el puerto serial (RS232) resulta muy simple e intuitiva. [32]

La creación de interfaces gráficas para el usuario (GUI), es muy básica y simplificada. Esta se puede realizar de una manera visual a través de Visual C#, arrastrando los elementos deseados a la interfaz, en lugar de hacerlo con código.

E. VISUALIZACIÓN EN 3D

1. GRÁFICOS 3D. En casi cualquier aplicación 3D se tienen los siguientes elementos:

- Un archivo que describe la geometría del mundo 3D. [48]
- Una cámara, la cual es el punto de visualización adentro del mundo. Esta cámara tiene como variables su localidad y la dirección. La imagen que se mira por la cámara es la que el usuario ve en la pantalla. [48]
- Objetos 3D adentro del mundo como personas, carros, aviones, etc. [48]

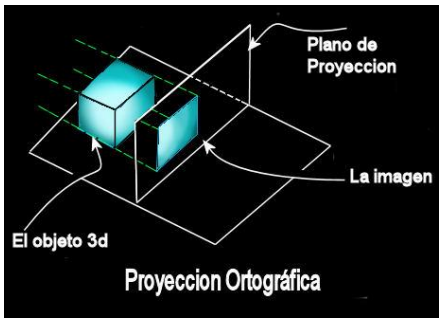
La tarea más importante del lenguaje 3D que se esté utilizando es calcular rápidamente la imagen que se va a ver por medio de la cámara, esta imagen depende del tipo de proyección que se está utilizando y del ángulo de la misma, a este proceso se le llama renderización. Cada vez que la cámara se mueve o un objeto dentro del mundo se mueve, se tiene que calcular nuevamente la imagen que se presentará frente a la cámara, para que se puedan observar los cambios en la pantalla. Al realizar este proceso varias veces por segundo, se genera la ilusión de un movimiento continuo, al cual se le llama animación 3D. [48]

La proyección es un método que se utiliza para engañar al ojo humano para que este pueda percibir objetos tridimensionales sobre un dispositivo bidimensional. Las dos técnicas más conocidas de para realizar una proyección, son la ortográfica y la perspectiva. [7] [28]

a. Proyección Ortográfica. Se trazan rayos perpendiculares al plano de proyección (son perpendiculares porque se considera la fuente de "luz" o centro de proyección en el infinito) y la imagen se forma con aquellos rayos que intersequen al objeto. [7] [28]

En la Figura No. 53 se puede observar una gráfica de la proyección ortográfica. Se puede notar, que esta proyección no deforma las figuras en la imagen resultante, debido a que utiliza rayos perpendiculares al plano de proyección.

FIGURA NO. 53. DIAGRAMA DE PROYECCIÓN ORTOGRÁFICA. [7]

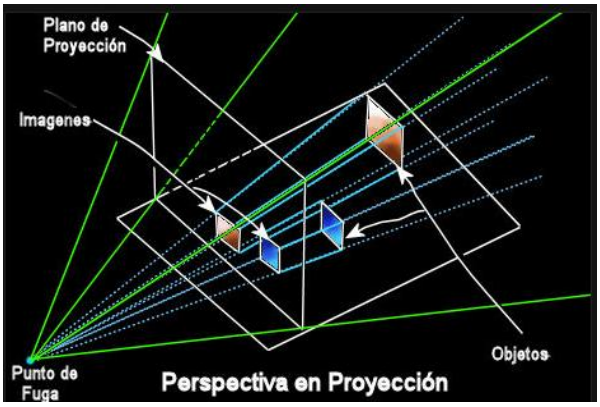


En este tipo de proyección no existe sensación de profundad, ya que variando la distancia del objeto al plano de proyección, no variará el tamaño de la imagen producida por este. Este tipo de proyección se utiliza en aplicaciones CAD, ya que permite ver el modelo desde diferentes partes sin que se deformen las dimensiones en la proyección. [7]

b. Proyección en perspectiva. En la proyección en perspectiva la fuente de rayos en este caso no se encuentra en el infinito, sino que es un punto localizado llamado punto de fuga. Los rayos que parten del punto de fuga e intersecan al objeto son entonces los que formaran la imagen sobre el plano de proyección. [7]

En la Figura No. 54 se puede observar una imagen de la proyección en perspectiva; se puede observar la deformación que se produce al utilizar este tipo de perspectiva.

FIGURA NO. 54. DIAGRAMA DE PROYECCIÓN DE PERSPECTIVA. [7]



En la proyección en perspectiva los objetos sufren una deformación por la distancia A medida que los objetos se alejan del plano de proyección “disminuyen” su tamaño de la misma manera en que ocurre cuando un objeto se aleja de nuestros ojos. Es por eso, que esta perspectiva es la más utilizada para dar una sensación de realismo, como la que se necesitan en los juegos, simulaciones y animaciones. [7]

En el caso de los lenguajes utilizados para hacer gráficos en las computadoras, tanto en el modo de perspectiva como en el ortográfico se limitan al espacio sobre el cual se realizan los cálculos, a esta región se le llama volumen de vista. Esta región es importante especificarla, ya que los objetos que se encuentran fuera de este campo, a derecha o a izquierda, no aparecerán en pantalla y los que se encuentren muy lejanos generarán pocos rasgos, por lo que de calcularse consumirían recursos innecesariamente. [7]

En la proyección ortográfica el volumen de vista tiene forma de caja rectangular y en la proyección en perspectiva, tiene forma de pirámide truncada con base rectangular. [7]

Ambos volúmenes se calculan a partir de la posición del observador, que en el caso de las aplicaciones en tres dimensiones el observador se encontraría en la posición de la cámara. [7]

La unidad básica de toda gráfica en tres dimensiones es el vértice. El vértice es un punto en el espacio, al cual se le puede agregar información como peso, color, posición, etc. [7]

2. LIBRERÍAS UTILIZADAS EN GRÁFICOS 3D. Todas las computadoras tienen hardware especializado para controlar los gráficos que se pueden ver en la pantalla, la función de las librerías utilizadas para hacer gráficos en 3D es decir a este hardware que hacer. Entre las librerías más utilizadas se encuentra OpenGl y DirectX. [48]

Toda aplicación 3D pasa gran parte del tiempo de ejecución diciéndole a las librerías que muevan los vértices de alguna manera específica. Todo este procesamiento

le llega al hardware de gráficos del computador, el es bastante bueno en procesar grandes sumas de vértices y en hacer polígonos a partir de ellos. El proceso de pasar de vértices a figuras en la pantalla se le llama “pipeline”. En este proceso se posicionan y se iluminan todos los vértices y además se hace la proyección configurada. Al final de este proceso todos los datos que se tenían de forma 3D se convierten a datos en 2D para que puedan ser presentados en la pantalla. [48]

a. OpenGl. OpenGl es una de las librerías más antiguas y más populares entre los desarrolladores de aplicaciones 3D. Nació en 1992 con la compañía Silicon Graphics Inc. Y ha crecido mucho desde entonces, ahora OpenGl se encuentra disponible para una gran variedad de plataformas, como lo son GameCube, Wii, PlayStation 3, iPhone, etc. [48]

La librería OpenGl tiene un estilo “C”, esto significa que en su lógica de programación no existen clases ni objetos, si no que se hace por medio de una larga colección de funciones. Internamente OpenGl es una máquina de estados, donde los llamados a las funciones alteran el estado del mismo, lo que a su vez afecta la respuesta del mismo.

Que OpenGl sea una máquina de estados, puede causar problemas si no se pensó todo de manera correcta, ya que se puede establecer cualquier estado en cualquier momento. [48]

b. DirectX. DirectX es una librería de gráficos desarrollada por Microsoft, creada para uso único de plataformas desarrolladas por ellos como Windows, Xbox, Windows Phone, etc. Una de las mayores ventajas de esta librería, es que se actualiza con bastante regularidad, por lo que siempre se tiene acceso a las últimas características gráficas creadas. El problema es que la compatibilidad se pierde mucho más rápido que con OpenGl, ya que con cada actualización, cambia la interfaz, y es necesario modificar el código para que funcione correctamente. Además de DirectX, Microsoft ha creado una librería para desarrolladores de aplicaciones 3D llamada XNA; esta utiliza DirectX en un nivel mucho más alto, lo que ayuda a desarrollar lo que se

desea en un tiempo más rápido, y además ayuda al nuevo desarrollador a aprender en un nivel más bajo de complejidad. [48]

1) **XNA.** Xna es un kit de desarrollo de software que provee un conjunto de componentes de programa prediseñados que se pueden usar como parte de código para elaborar otros programas. XNA específicamente provee código de programación para dibujar en pantalla, producir sonidos, etc.

El objetivo de cualquier kit de desarrollo de software y de XNA es hacer más fácil el entendimiento del control de los gráficos de la computadora.

El entorno de trabajo para el que fue creado esta librería es Visual Studio es por esto que se decidió programar la aplicación 3D con Microsoft Visual C#. [48]

2) **Requerimientos del sistema.** En este proyecto se trabajó sobre Microsoft Visual C# 2010 Express Edition y con el entorno de desarrollo integrado (IDE) XNA 4.0. Estos dos recursos son software completamente libre, es decir ambos son gratis y abiertos a descargas para cualquier persona. Para instalar XNA Game Studio 4.0 es necesario primero instalar Visual Studio 2010 edición estándar o alguna más reciente. [43]

3. **DESARROLLO 3D.** El sistema de coordenadas en el desarrollo de programas con gráficos en 3D es llamado “mundo”; el centro de este se encuentra en el origen, en la coordenada (0, 0, 0). Cuando se trabaja en 3D hay que tener en cuenta muchas variables para que el objeto que se quiera dibujar aparezca en donde queremos que aparezca, ya que a diferencia con gráficos en 2D donde sólo se le dice una posición en pantalla, en 3D es necesario que se diga la posición del gráfico en el mundo, y la posición de la cámara, es decir, en gráficos 3D la posición del objeto puede variar al mover su posición en el mundo o al mover la posición o ángulo de vista de la cámara. [34] [48]

Al desarrollar gráficos en 3D es necesario conocer el sistema de coordenadas 3D y los movimientos que se consideran positivos para cada uno de los ejes. Generalmente el

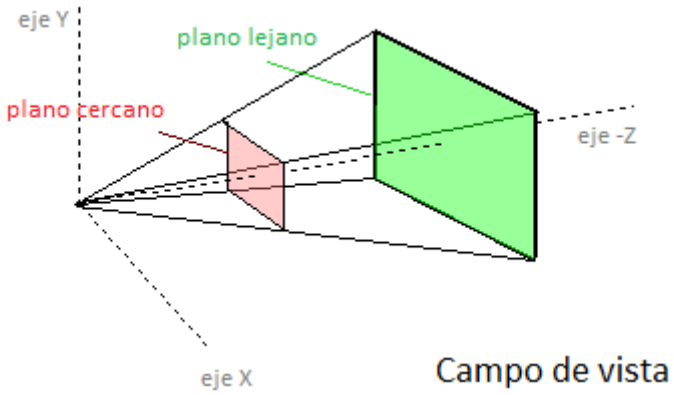
eje X se mueve positivamente hacia la derecha y el Y se mueve positivamente para arriba. Para la coordenada Z no se tiene un movimiento positivo completamente definido, por lo que se puede trabajar con cualquiera de los dos, cada uno con diferente nombre: sistema de coordenadas de mano izquierda o sistema de coordenadas de mano derecha. Estos se llaman de esta manera, porque el pulgar apunta hacia el movimiento de Z positivo en el sistema de coordenadas. En XNA se utiliza un sistema de coordenadas de mano derecha, por lo que se avanza positivamente en Z cuando se observa que el movimiento se acerca hacia la pantalla. [34] [48]

a. Cámara 3D. Para definir una cámara en el mundo 3D las primeras características que se definen son la vista y la proyección, ambas características se definen mediante matrices. La vista de la cámara almacena información sobre la posición de la misma en el mundo, la dirección hacia donde está apuntando y su orientación y la matriz de proyección almacena las propiedades basadas en el ángulo de la vista, que tan lejos puede ver, etc. Esta última matriz representa la transformación de un mundo 3D a la pantalla plana en 2D. [43]

Para crear la matriz de vista XNA provee un método llamado CreateLookAt, este método recibe tres vectores de tres dimensiones, el primero es la coordenada de la posición de la cámara, el segundo la coordenada del punto hacia donde la cámara está viendo y el tercero el vector que indica la donde se encuentra la posición de arriba con respecto a la cámara. [43]

Para crear la matriz de proyección también existe otro método llamado CreatePerspectiveFieldOfView; este método recibe el ángulo de vista permitido de la cámara, la relación de aspecto, es decir la relación entre el ancho y la altura de la pantalla, el plano cercano de vista, el cual define que tan cerca puede estar un objeto antes de dejarse de ver y el plano lejano de vista, el cual define que tan lejos de la cámara puede estar un objeto antes de dejarse de ver. Todos estos parámetros le sirven a la cámara para definir un área de visibilidad para la cámara. [43]

FIGURA NO. 55. CAMPO DE VISTA DE LA CÁMARA. [26]



En la Figura No. 55 se puede ver gráficamente el campo de vista que se crea al definir la matriz de proyección; todos los objetos que se encuentran en ésta área son dibujados. El ángulo de apertura de este triángulo corresponde al ángulo de vista y la forma del plano lejano como la del cercano corresponden a la relación de aspecto. [43]

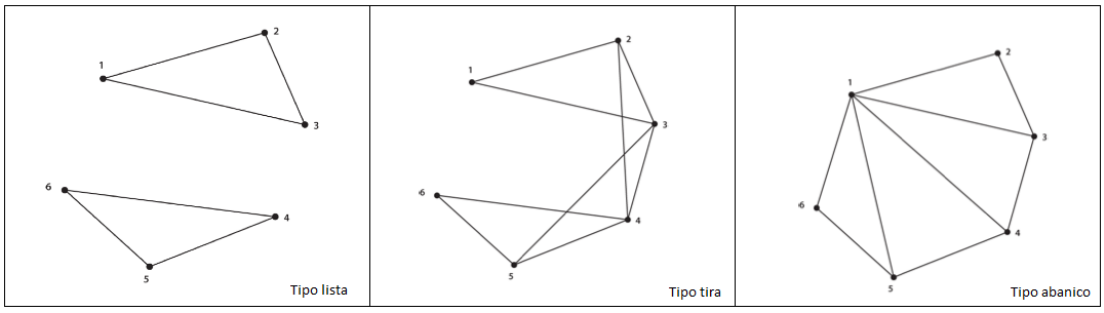
b. Figuras primitivas. La base de todas las figuras que son dibujadas en 3D es un triángulo, si se dibujan los suficientes se puede obtener casi cualquier figura posible, si se ponen lo suficiente cerca y pequeños se puede obtener una superficie bastante suave y hasta redonda. Esto es bastante parecido a la realidad, ya que si se mira cualquier superficie por medio de un microscopio se puede ver que está formada por pequeños formas rígidas, el nivel de detalle de una figura, depende entonces de la cantidad de triángulos que puedan haber en la misma, siendo una limitante la capacidad de procesamiento de la computadora que estamos utilizando. [43]

Para dibujar un triángulo en XNA es necesario definir sus vértices, crear un buffer de vértices, el cual sirve para almacenar toda la información de los gráficos a dibujar y crear un efecto para dibujar las primitivas; en este caso se utilizó un efecto que permite dar la información de posición y color a cada vértice que se dibuja. Cuando se tiene toda la información que se desea dibujar en el buffer de vértices se llama a un método para dibujar las primitivas, el cual se encarga de enviar la información hacia el dispositivo de gráficos de la computadora y de decirle que tipo de información es. En este caso le diría que le va a enviar vértices cada uno con su posición y su color. [43]

Todo en XNA se dibuja por medio de un lenguaje de sombras de alto nivel, es por esto que se necesita instanciar un efecto, ya que este nos permite comunicarnos con este lenguaje. XNA provee una clase derivada de la clase efecto llamada “BasicEffect”; esta clase nos permite utilizar el lenguaje de sombras de alto nivel, sin necesidad de conocer algo del mismo. Para utilizarlo, primero debemos instanciar algunas propiedades del mismo como lo son la propiedad mundo, la propiedad de vista y la propiedad de proyección. Las propiedades de vista y de proyección son iguales a las que se instanciaron previamente en la cámara y la de mundo debe ser la matriz identidad ya que se está dibujando con respecto al origen, por lo que no se le debe de aplicar ninguna transformación al mismo. [43]

1) **Tipos de primitivas.** Existen tres diferentes formas de dibujar primitivas de triángulos en XNA, una lista, una tira o un abanico. La más básica de las tres es la lista, ya que al recibir tres vértices dibuja un triángulo entre ellos, esta es bastante fácil de utilizar pero también es la menos eficiente, ya que se deben de especificar tres vértices nuevos por cada triángulo dibujado. Al dibujar con tira, se dibuja un triángulo con los tres primeros vértices pero dibuja otro al agregar un nuevo vértice, este último triángulo se dibuja utilizando los últimos dos vértices especificados para el triángulo anterior. Y en el último tipo (abanico) se dibuja el primer triángulo con los primeros tres vértices y los siguientes se utiliza un nuevo punto que se le dé, uno anterior y el primero, por lo que se forma una especie de abanico. En la Figura No. 56 se pueden estas diferentes formas de dibujo para los triángulos como primitivas. [43]

FIGURA NO. 56. TIPOS DE FORMAS DE DIBUJO DE PRIMITIVAS. [34]



c. Movimiento y rotación. Todo lo que se dibuja en XNA se puede rotar o trasladar. Una traslación mueve un objeto en la dirección que se especifique mediante un vector, por lo que en XNA existe un método llamado crear traslación ("CreateTranslation") el cual recibe un vector de tres dimensiones como parámetro. [43]

Para realizar una rotación es necesario especificar el eje en el cual se va a dar la rotación, en XNA existe un método para realizar rotaciones en cada uno de los ejes X, Y, y Z el cual recibe el ángulo en radianes a rotar. [43]

Estas traslaciones y rotaciones mueven a los objetos no a la cámara, por lo que son propiedades individuales para cada primitiva que se dibuje, la traslación se hace con base a la posición actual de la primitiva y la rotación en base a el eje X, Y, o Z de la primitiva. [43]

Algo muy importante en el movimiento de las primitivas es el orden en que se haga la multiplicación de las matrices de traslación y de rotación; si primero se hace la traslación y después la rotación, el objeto al que se le aplique esta transformación siempre rotará alrededor del origen, pero si primero se multiplica por la traslación y después por la rotación, la primitiva siempre rotará alrededor de su propio eje, aunque se traslade a cualquier otro lado. Con estas dos propiedades se puede crear casi cualquier movimiento deseado, por ejemplo, si se quiere hacer un movimiento como el de la tierra alrededor del sol (girando sobre su propio eje y orbitando alrededor del sol), se puede realizar aplicando primero una rotación para que la primitiva gire en su propio eje, después una traslación y después otra rotación, para hacer el efecto que se tuvo inicialmente al hacer primero la traslación y después la rotación, que gire alrededor de su origen. [43]

Los movimientos anteriores sirven para mover a las figuras que se crean dentro del mundo, pero para simular un mejor movimiento sería más lógico mover solamente la cámara y dejar estáticas todas las figuras primitivas. [43]

Para crear una cámara que se pueda mover en el espacio 3D, es necesario modificar la matriz de vista de la misma. La matriz de vista contiene información sobre la posición y rotación de la cámara. Específicamente contiene un vector de posición, un

vector hacia dónde la cámara se encuentra viendo y un vector que dice donde es arriba. [43]

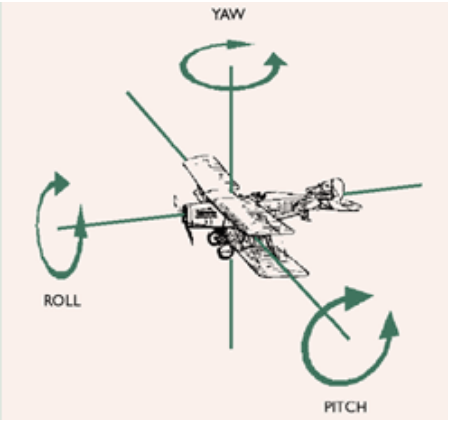
Para el movimiento es necesario tener una variable que represente la dirección relativa hacia donde la cámara está viendo, esta variable se puede definir como la resta normalizada entre el vector que define hacia donde se encuentra viendo la cámara y el vector de la posición de la misma. Esta resta se debe normalizar para convertir el vector resultante en un vector con magnitud de 1, lo cual va a ser útil para mover la cámara en una dirección específica. [43]

En este momento se requiere que la cámara se pueda mover en la dirección hacia donde esta se encuentra viendo. Con el vector de dirección de la cámara normalizado, el movimiento se vuelve bastante simple, ya que sólo es necesario sumar este vector al vector de posición y multiplicarlo por un factor, el cual podría verse como la velocidad a la que se mueve la cámara. [43]

Para mover la cámara en un movimiento perpendicular al anterior se necesita un vector perpendicular al vector de dirección de la cámara. Para obtenerlo es necesario hacer el producto cruz entre el vector de dirección normal y el vector que dice donde es arriba. Entonces para mover la cámara hacia los lados sólo es necesario, como en el caso anterior, sumar el vector resultante del producto cruz, a la posición actual de la cámara.

Existen tres rotaciones que un objeto puede realizar en tres dimensiones: “yaw”, “pitch” y “roll”. [43] Estas rotaciones se puede observar en la Figura No. 57 .

FIGURA NO. 57. POSIBLES ROTACIONES DE UN OBJETO 3D. [26]



Estas rotaciones no corresponden necesariamente a las rotaciones alrededor de los ejes X, Y y Z, debido a que en el momento en que la cámara rota en cualquiera de estas tres direcciones, las otras dos ya no se encuentran exactamente en los ejes. Ésta es la razón por la cual las rotaciones se definen en base a la posición actual del objeto que se desea rotar. En este caso para hacer una rotación de tipo "yaw", se puede utilizar el vector de la cámara que dice donde es arriba como eje de rotación. Para hacer la rotación de tipo "roll" se utiliza el vector de dirección de la cámara y para la de tipo "pitch" se utiliza el vector perpendicular al vector de dirección.

Para implementar la rotación tipo "yaw", aunque esta gira alrededor del vector de la cámara que dice donde es arriba, el vector que realmente se ve afectado es el vector de dirección de la misma. XNA provee una función llamada "transform" la cual obtiene el vector afectado, en este caso el vector de dirección y una matriz que representa la rotación o traslación que se le quiere aplicar al vector. Para crear la matriz que representa la rotación, XNA también provee algo que facilita el proceso, el método "CreateFromAxisAngle". Este método crea una rotación alrededor del vector que se le especifique y la cantidad de grados que se le especifique.

Con estos dos métodos fácilmente se puede implementar cualquier rotación, sólo se debe de tener claro, que vector se ve afectado en cada caso y el vector sobre el cual se da la rotación. En el caso de "yaw" el vector afectado es el de dirección y el eje de rotación es el vector que le dice donde es arriba a la cámara. Para crear un "roll" el vector afectado es el que dice donde es arriba y el eje de rotación es el vector de dirección. Y por último para crear una rotación de tipo "pitch" se debe hacer un proceso un poco más complicado, ya que este movimiento afecta tanto al vector de dirección como al vector que dice donde es arriba. Sin embargo las dos transformaciones que se deben de hacer dependen de la misma matriz de rotación la cual tiene como eje el vector perpendicular al de dirección. [43]

d. Sacrificio de cara posterior. El sacrificio de cara posterior ("backface culling") es un proceso que se trabaja en XNA para limitar el número de objetos que son dibujados en una pantalla y así mejorar el desempeño; el objetivo es

dibujar sólo la parte de la primitiva que se está frente a la cámara. Por defecto XNA sacrifica todas las primitivas que hayan dibujado sus vértices en sentido contrario al de las agujas del reloj. Esta opción puede ser opcional, pero no es muy recomendable quitarla ya que se puede perder rendimiento. [43]

e. Modelos 3D. Hoy en día casi ninguna aplicación sigue utilizando figuras primitivas para dibujar lo que necesite, ya que es un proceso muy largo que puede consumir mucho tiempo al dispositivo de gráficas del computador utilizado, en figuras muy complicadas, es por esto que existen los "Modelos 3D"; los modelos al igual que las primitivas están compuestos por una serie de puntos, colores y texturas, estos modelos generalmente se crean fuera de XNA en aplicaciones especializadas como MAYA, Blender, 3D Studio, etc. Los formatos utilizados por XNA para modelos 3D son .X y .FBX. [43]

IV. ANTECEDENTES

El megaproyecto SEEQ, no tiene ningún precedente en la Universidad del Valle de Guatemala. Aunque se podría pensar que el primer megaproyecto realizado en la Universidad llamado *Diseño y construcción de un sistema explorador robotizado y autárquico* [12] está relacionado de cierta manera, pues el objetivo esencial de ambos es explorar ambientes, el megaproyecto SEEQ se diferencia en diversos aspectos. El primero es que el vehículo a utilizar para realizar la construcción de mapas y la exploración es un vehículo aéreo. El segundo es que la técnica utilizada para recaudar información del ambiente es diferente, basándose la original en una cámara, mientras que la trabajada se basa en un sensor láser. Otra diferencia es que el proyecto anterior no genera un mapa, simplemente busca movilizarse en el ambiente evadiendo obstáculos, mientras que es uno de los objetivos primordiales del megaproyecto SEEQ generar un mapa completo del ambiente explorado.

A pesar que en el país no existen antecedentes significativos para SEEQ, existen muchos estudios internacionales muy similares, los cuales se utilizaron como base y guía para el megaproyecto. El estudio realizado por estudiantes de la universidad de MIT titulado *Autonomous Navigation and Exploration of a Quadrotor Helicopter in GPS-denied Indoor Environments* sirvió de motivación para llevar a cabo el megaproyecto. Este trabajo presenta una solución para que un helicóptero con cuatro rotores pueda navegar autónomamente, explorar y localizar objetos de interés en ambientes cerrados y desconocidos. El objetivo principal de este trabajo fue crear un vehículo aéreo completamente autónomo, que pueda entrar a un ambiente por medio de una ventana, explorar un ambiente desconocido, buscar un objeto de interés y transmitir un video del objeto localizado. Los retos del proyecto fueron, diseñar un helicóptero con cuatro rotores que pueda estabilizarse autónomamente, y desarrollar un algoritmo que pueda navegar en ambientes sin GPS. [1]

De este proyecto se tomaron las ideas iniciales para SEEQ, pero debido a la falta de recursos, se obviaron algunos aspectos del mismo. En la Figura No. 58, Figura No. 59, y Figura No. 60, se pueden observar imágenes de los resultados del proyecto *Autonomous Navigation and Exploration of a Quadrotor Helicopter in GPS-denied Indoor Environments.* [1]

FIGURA NO. 58. MAPA DEL PISO UNO DE LA UNIVERSIDAD MIT. [1]

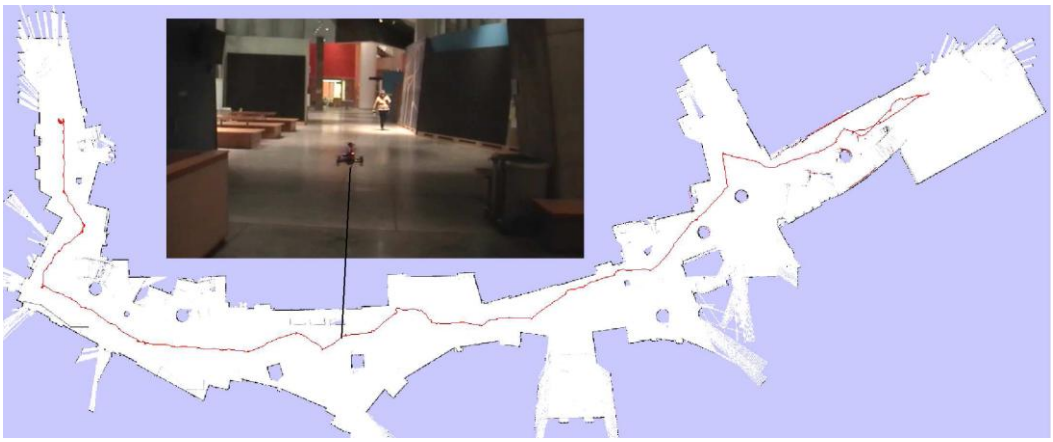


FIGURA NO. 59. MAPA DEL TERCER NIVEL DE LA UNIVERSIDAD MIT. [1]

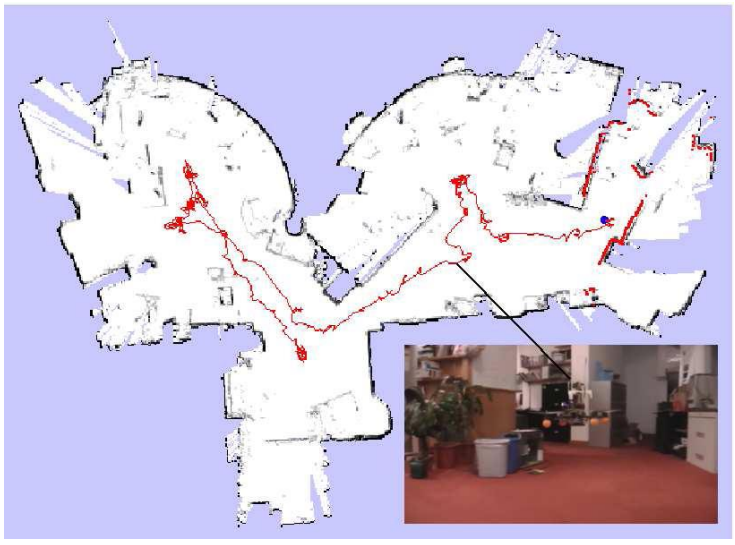
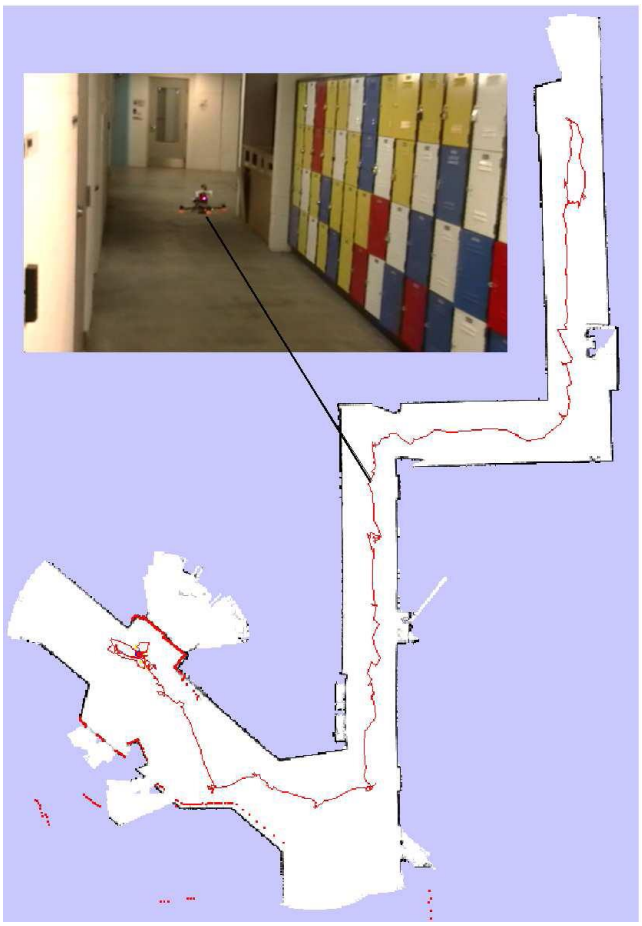


FIGURA NO. 60. MAPA DEL SOTANO DE LA UNIVERSIDAD MIT. [1]

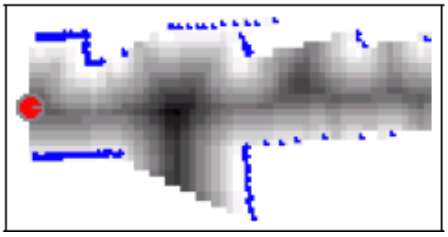


Otro trabajo que tuvo mucha influencia en el megaproyecto SEEQ fue el que se realizó en las universidades Carnegie Mellon University en Pittsburgh, PA, y la University of Freiburg, en Alemania, titulado "*A Real-Time Algorithm for Mobile Robot Mapping With Applications to Multi-Robot and 3D Mapping*" [60]. Este proyecto presenta una solución que realiza un mapeo en tiempo real de una forma autónoma. El enfoque de este trabajo hacia el problema de exploración y mapeo simultáneo es un poco más conservador que el anterior, pues trabaja solamente con vehículos terrestres. Este proyecto ayudó bastante, porque el megaproyecto SEEQ, aunque es un helicóptero, y se moviliza por el medio aéreo, el mapeo que realiza es a una altura determinada, por lo que el mapeo termina siendo similar al de este proyecto, en dos dimensiones.

En el estudio, para realizar el mapeo, se basan en la posición del robot al momento de realizar el escaneado con los sensores, y de la velocidad del mismo. El

algoritmo que obtiene estas entradas, da como respuesta el mapa más probable, el cual es representado gráficamente en una etapa posterior. [60]

FIGURA NO. 61. MAPA DE PROBABILIDAD GENERADO POR UN ESCANEO. [60]



El algoritmo utilizado en la publicación es denominado por sus autores como mapeo incremental. De una manera simplificada, el algoritmo consiste en obtener datos de los sensores, calcular la posición del robot, adjuntar la posición y el escaneo a un mapa global, y congelarlo. [60]

El problema con esta técnica, es que no se tienen ninguna defensiva contra el error en el mapa, y como no se realiza un ajuste al incrementar el mapa global, el error se va acumulando, teniendo como resultado un mapa inexacto. Para poder utilizarla el vehículo y los sensores deben de tener una incertidumbre muy baja. Esto no se pudo aplicar por completo al helicóptero ya que los movimientos inexactos del mismo producen un error de posición, el cual debe de ser eliminado en el mapa, para obtener una buena representación del ambiente.

V. DELIMITACIÓN E IMPACTO DEL TEMA

El megaproyecto SEEQ Robohelicóptero pretende, desarrollar un vehículo que domine el medio aéreo, con fines de exploración, generación y visualización de mapas en ambientes desconocidos. La idea inicial fue realizar una exploración completa de todo el lugar en donde se encuentre. A falta de recursos, la exploración realizada se llevó a cabo a una altura determinada; esto afecta directamente a la generación y visualización del mapa, ya que aunque se tiene una visualización de tres dimensiones, los datos que se utilizaron para generarla fueron datos obtenidos a una altura específica.

El proceso del megaproyecto empieza desde diseñar y construir la estructura del helicóptero. El diseño se basó en un helicóptero de cuatro rotores el cual considera los esfuerzos, vibraciones y deflexiones que podrían afectar a la estructura. También se hicieron los análisis necesarios para seleccionar los motores y las hélices apropiadas para levantar el peso de la estructura.

Este proyecto también abarcó el área de vuelo y control. Esta parte del proyecto se concentró en realizar un vuelo estable midiendo las variables de posicionamiento del helicóptero en el ambiente. Esta estabilización es muy importante, ya que es la que permite que el helicóptero pueda completar las instrucciones de vuelo que reciba.

Para realizar el mapa de tres dimensiones se utilizó un sensor láser. Esta parte del proyecto se limitó a tomar datos y a comunicarlos por medio de radiofrecuencia al módulo de mapeo y exploración (vuelo automatizado). En este último módulo se realiza la recepción de los datos obtenidos del sensor láser, el procesamiento y análisis de los mismos y finalmente el envío de datos al módulo de visualización.

El módulo de mapeo y exploración, recibe datos del sensor a una altura determinada, es decir, a la altura que vuele el helicóptero. Esto significa que el análisis de los datos obtenidos, se limita a un análisis de dos dimensiones. Los datos que se reciben en este módulo, se utilizan para la construcción incremental del mapa del ambiente explorado. Este mapa es de gran importancia, ya que los datos de posición y distancia

entre paredes retroalimentan al helicóptero para que su vuelo sea más estable y seguro. Además este mapa es el que utiliza el módulo de visualización para presentarle al usuario los datos que se han recopilado durante la exploración.

Otra tarea importante del módulo de mapeo y exploración es la planificación y toma de decisiones para el movimiento del robot. En este punto se analiza el mapa en construcción y se le indica al robot la siguiente posición a la que se debe movilizar; esto con el objetivo de realizar la exploración de una manera eficiente.

Finalmente el módulo de mapeo en tres dimensiones toma los datos del módulo de vuelo automatizado y los grafica en tres dimensiones dándole una altura específica a cada punto. Este mapa resulta bastante representativo del lugar que se escanea, y se puede ver desde dos vista, una de planta y una de interior. Esto se realiza con el objetivo de presentarle al usuario una forma bastante comprensiva e interactiva de visualizar los datos. Este programa se realizó utilizando lenguaje C# y las librerías de XNA de DirectX. Se piensa que este módulo será útil para futuros proyectos con necesidad de mapeo, ya que se utilizaron recursos libres de descarga.

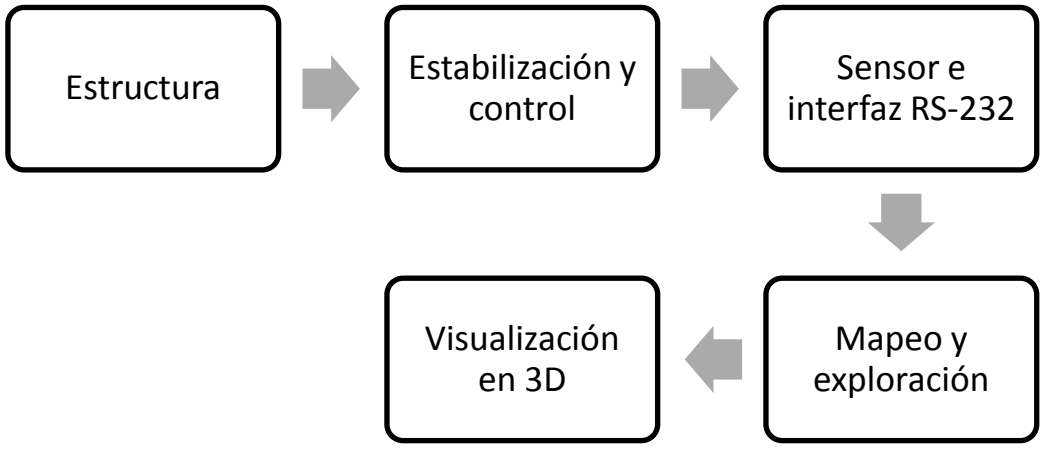
De la misma manera en que el robot de éste proyecto analiza mapas en 2D, se podría realizar un análisis completo de un ambiente dado en 3D, con la incorporación de otros tipos de sensores auxiliares al sistema. Esta etapa de desarrollo queda fuera del alcance de este trabajo, pero se recomienda como una siguiente fase de seguimiento al proyecto.

VI. METODOLOGÍA

El megaproyecto SEEQ, se modularizó en cinco áreas de trabajo, las cuales se fueron trabajando en paralelo durante los tres semestres que duro el megaproyecto. Sin embargo, estos módulos interactúan de una manera específica, y es por esta interacción y codependencia que se estructura el presente reporte en un orden determinado. Este orden no es cronológico, pero muestra de una manera clara la relación entre módulos.

En el diagrama de bloques de la Figura No. 62 se muestran los cinco módulos a desarrollar, y el orden en el que se presentan. Cada uno de los módulos manejó su propia metodología de trabajo, y esta se detalla en el capítulo correspondiente a cada tema.

FIGURA NO. 62. DIAGRAMA DE BLOQUES DEL MEGAPROYECTO SEEQ



VII. ESTRUCTURA

A. MODELADO TRIDIMENSIONAL DEL SISTEMA

1. DISEÑO EXPERIMENTAL. Se utilizó el software de Autodesk Inventor para realizar el dibujo tridimensional. En esta etapa se desarrolla un diseño de cuatro bases para rotores colocados en los extremos de una cruz, que forma la base central o el esqueleto de nuestro helicóptero y que es cubierta parcialmente por una cavidad para almacenamiento de baterías y componentes electrónicos. Se realiza una pequeña cruz inicial y se ensamblan cuatro tubos circulares sobre ella para poder generar la base principal del cuadcóptero.

2. RESULTADOS

FIGURA NO. 63. BANCADA Y BRAZO PARA SOSTÉN DE LOS CUATRO MOTORES

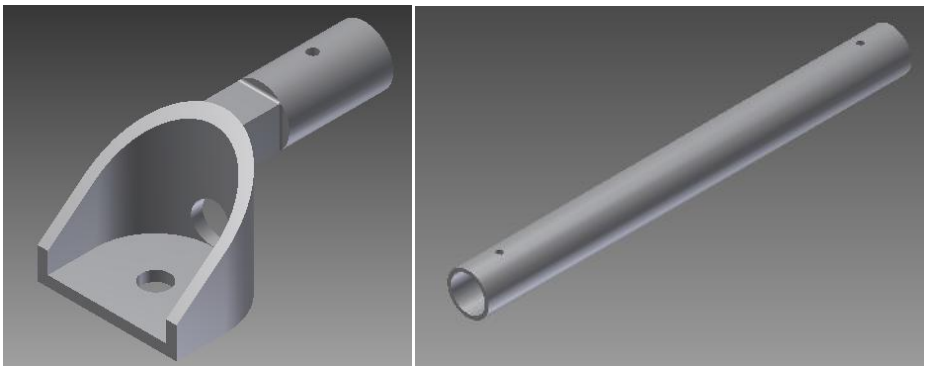


FIGURA NO. 64. PARACHOQUES Y CRUZ CENTRAL PARA ENSAMBLE DE LOS CUATRO BRAZOS

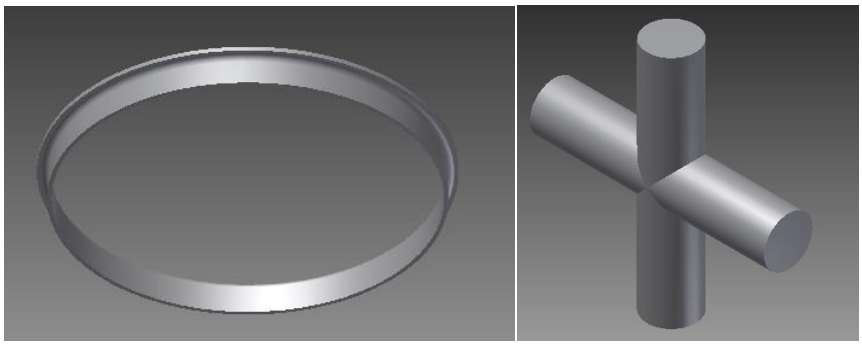


FIGURA NO. 65. CAJA CENTRAL PARA COMPONENTES ELECTRÓNICOS

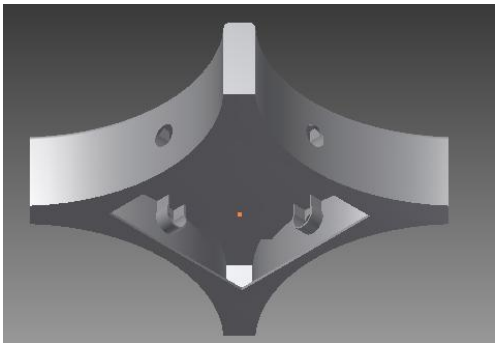
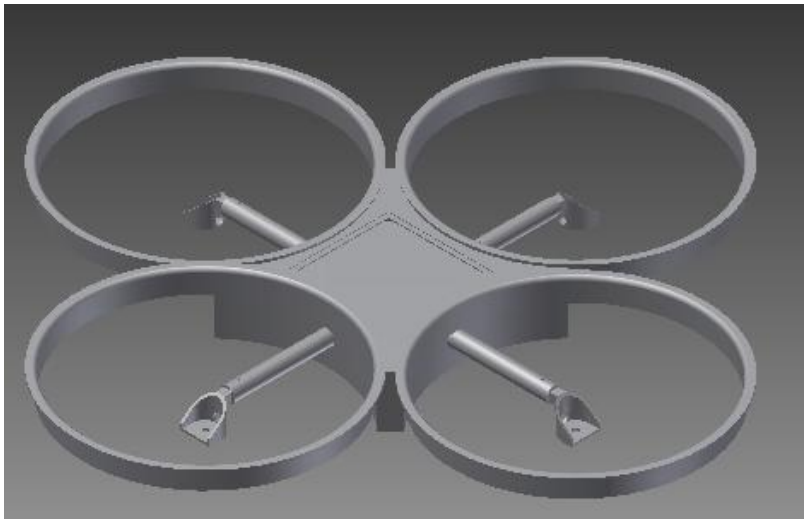


FIGURA NO. 66. MODELO TRIDIMENSIONAL DEL QUADCÓPTERO



3. DISCUSIÓN. De este modelo se puede comenzar a especular sobre aspectos estéticos y geométricos para poder cambiar ciertas formas mientras todavía es un dibujo y no cuando ya se ha implementado en vida real.

B. SELECCIÓN DE HÉLICE Y POTENCIA DE MOTORES.

1. DISEÑO EXPERIMENTAL. Para esta etapa se seleccionó un perfil de hélice, por medio de su curva característica se tomó lectura de las revoluciones por minuto a las que cada hélice debe girar para levantar 1.1 kilogramos, se obtuvo el valor de torque requerido a esas revoluciones y se calculó la potencia mecánica requerida.

Se compró el motor apropiado con un controlador capaz de suministrar la potencia calculada, tomando en cuenta una eficiencia del motor de 65%.

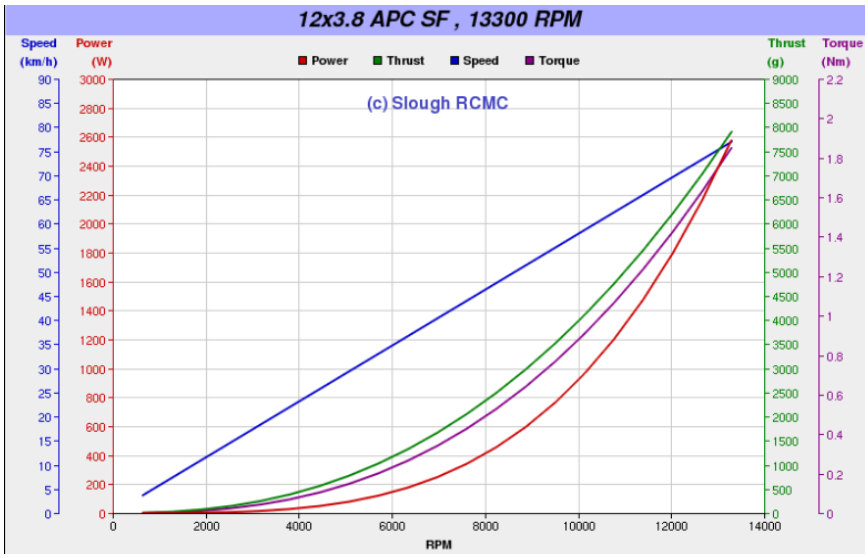
2. CÁLCULOS Y RESULTADOS. Para el trabajo presentado en este documento, se desea levantar aproximadamente cuatro kilogramos aún cuando la masa real es menor. La razón es porque se diseñará la estructura en caso se desea agregar peso extra posteriormente. La aceleración de diseño será 1 m/s2, de esta manera la sustentación por cada motor, asumiendo que el peso se distribuye uniformemente con 1 Kilogramo por motor, será de 11 Newton. Lo cual se calcula con la siguiente ecuación.

FIGURA NO. 67. ECUACIÓN DE FUERZA DE SUSTENTACIÓN

$$Sustentación = peso + (masa * aceleración)$$

Para nuestro diseño se utiliza un modelo estándar conocido como APC 12x3.8.

FIGURA NO. 68. CURVA CARACTERÍSTICA DE LA HÉLICE APC 12X3.8



Una vez establecida la hélice a utilizar se procede entonces a seleccionar un motor con la potencia necesaria para llevar a la hélice a las revoluciones necesarias, que permitan la sustentación requerida y a la vez sea capaz de vencer el arrastre. La sustentación requerida para nuestro diseño es de 11 Newton. De acuerdo con la curva característica para la hélice, se puede observar que para generar una fuerza de 11 Newton (1.1 Kilogramos) se necesita mover la hélice aproximadamente a 5700 RPM, lo que a su

vez exige que se proporcione 0.2 Nm de torque. Con esta información se puede obtener entonces la potencia mecánica requerida. Mediante la siguiente ecuación.

FIGURA NO. 69. ECUACIÓN PARA POTENCIA MECÁNICA REQUERIDA

$$\mathbf{P = \frac{2\pi * rpm * Torque}{60}}$$

$$P = 119.38 \text{ watts}$$

Para nuestro diseño se implementaron cuatro motores DC sin escobillas Turnigy 2217 20turn Outrunner. Las especificaciones se muestran en la Figura No. 70.

FIGURA NO. 70. CARACTERÍSTICAS DE MOTOR TURNIGY 2217 20TURN OUTRUNNER. [20]

Kv (rpm/v)	860
Weight (g)	71
Max Current (A)	17
Resistance (mh)	0
Max Voltage (V)	11
Power(W)	0
Shaft A (mm)	-
Length B (mm)	36
Diameter C (mm)	28
Can Length D (mm)	21
Total Length E (mm)	51

Para el cálculo de la potencia del motor se utilizó la siguiente relación asumiendo una eficiencia de 65% para motores DC sin escobillas.

FIGURA NO. 71. VALOR DE POTENCIA DEL MOTOR

$$P = 0.65VI$$

$$P = 121.55 \text{ Watts}$$

Para el control del motor DC sin escobillas, se utilizó un controlador capaz de soportar el amperaje consumido, este es el recomendado por el fabricante, un Turnigy Plush 25Amp Speed controller. Este dispositivo ayuda obtener una función lineal entre PWM y la fuerza de sustentación. Permite controlar la velocidad del motor mediante una señal PWM. Sus especificaciones se muestran en la Figura No. 72.

FIGURA NO. 72. CARACTERÍSTICAS DE CONTROLADOR DE MOTOR TURNIGY PLUSH 25 A. [22]

Cont Current:	25A
Burst Current:	35A
BEC Mode:	Linear
BEC :	5v / 2A
Weight:	22g

Finalmente, se selecciona una batería en función del tiempo de vuelo y del peso de la misma. Dado que se utilizarán dos baterías y cada motor consume como máximo 17 amperios, se selecciona una batería de 5Ah. El tiempo de vuelo se calcula de la siguiente forma considerando que los motores estuviesen al máximo siempre.

FIGURA NO. 73. DEDUCCIÓN DE TIEMPO DE VUELO EN MÁXIMA POTENCIA

$$\textbf{Tiempo} = \frac{\textbf{Amperios Hora} * \textbf{60}}{\textbf{Amperios consumidos en dos motores}}$$

$$\text{Tiempo} = 8.823 \text{ minutos}$$

La batería seleccionada es una Turnigy 5000mAh de 20C Lipo Pack. Cuyas características se muestran en la Figura No. 74.

FIGURA NO. 74. CARACTERÍSTICAS DE BATERÍA TURNIGY 5000MAH DE 20C LIPO PACK. [21]

Capacity(mAh)	5000
Config(s)	3
Discharge(c)	20
Weight(g)	412
Max Charge Rate (C)	2
Length-A(mm)	145
Height-B(mm)	49
Width-C(mm)	26

3. DISCUSIÓN. La hélice para nuestro diseño pudo haber sido otra, ya que dependiendo del motor se podría generar la misma sustentación, la diferencia es que debería ir a mayores revoluciones por minuto y además presentaría un distinto torque para el motor. Sin embargo, los fabricantes de hélices pequeñas presentan gráficas con menor rango de sustentación permitida. Por esta razón se utilizan hélices lo suficientemente grandes capaces de levantar mayores cargas.

La elección de los demás componentes de potencia (motores, controladores y baterías) están en función de la potencia mecánica requerida y tomando en cuenta una eficiencia de 65% para motores DC sin escobillas.

C. DISEÑO CONTRA VIBRACIONES

1. DISEÑO EXPERIMENTAL. Una vez seleccionado el motor y una hélice necesaria para generar la sustentación necesaria, se procedió a realizar un análisis crítico en el diseño del helicóptero, este es el diseño de vibración. El objetivo fue establecer un grosor y un perfil adecuado para cada uno de los tubos que sostienen a los motores de manera que no exista vibración excesiva y principalmente evitar que el rango de frecuencias utilizadas en las revoluciones del motor, estén cerca de la resonancia del sistema que forman los tubos. Dado que la magnitud de la fuerza de desbalance de la hélice se desconoce, simplemente se realizó una comparación de las frecuencias en las que el sistema opera y la frecuencia natural del sistema para establecer un criterio de 0.7071 entre el cociente de la frecuencia máxima utilizada y la frecuencia de resonancia.

2. CÁLCULOS Y RESULTADOS. El perfil deseable para los brazos del helicóptero es un perfil de aluminio circular, hueco por dentro. La fuerza de desbalance produce vibración especialmente por flexión en la viga y por torsión, de esta cuenta se utilizan los siguientes modelos matemáticos para determinar la constante de resorte lineal y torsional.

FIGURA NO. 75. CONSTANTES DE RESORTE

$$k = \frac{3EI}{L^3}$$

$$kt = \frac{GI_p}{L}$$

De donde para un perfil circular hueco,

FIGURA NO. 76. INERCIA

$$I = \frac{\pi(dext^4 - dint^4)}{64}$$

$$I_p = \frac{\pi(dext^4 - dint^4)}{32}$$

Siendo E el módulo de elasticidad de Young, G el modulo de rigidez, L la longitud del brazo hacia donde se encuentra el motor, Ip la inercia polar, I el primer momento del área, dext el diámetro externo y dint el diámetro interno del perfil. Para obtener la frecuencia angular natural del sistema en flexión y torsión, se utilizaron las siguientes relaciones respectivamente.

FIGURA NO. 77. FRECUENCIA NATURAL

$$\omega n = \sqrt{\frac{k}{m}}$$

$$\omega nt = \sqrt{\frac{kt}{J}}$$

Donde J es la inercia de la masa con respecto al eje de rotación modelada como un pequeño cilindro que rota en su parte más alta. Para ello se utilizó el siguiente modelo matemático.

FIGURA NO. 78. INERCIA CON RESPECTO AL EJE DE ROTACIÓN

$$J = \frac{\text{Masa} * \text{Altura del cilindro}^2}{3}$$

Los datos para las ecuaciones se muestran en la Tabla No. 3.

TABLA NO. 3. DATOS INICIALES PARA CÁLCULOS DE VIBRACIÓN

Característica	**Valor**	**Unidades**
Longitud de brazo	160	mm
G aluminio	269	Gpa
E aluminio	717	Gpa
Masa (motor + bancada)	0.1	Kg
ω de operación	5700	rpm
Altura del motor	36	mm
J (inercia motor)	0.0000432	Kgmts^2

Los perfiles de aluminio más pequeños que se venden en Guatemala son de ½,¾ y 1 de pulgada de diámetro, con espesores de 1/32 y 1/16 pulgada. En las Tablas No. 4-6, se muestran los resultados de inercia, constante de resorte, frecuencia natural, frecuencia de operación y relación de frecuencias para diversos perfiles de aluminio sometidos a flexión y torsión.

TABLA NO. 4. PERFIL CIRCULAR DE 1/2 PULG DE DIÁMETRO Y 1/32 PULG DE ESPESOR

Flexión			**Torsión**		
I	5.28439E-10	mts^4	**Ip**	1.05688E-09	mts^4
k	27750.76754	N/m	**Kt**	177.6874808	Nm
ωn	526.7899728	rad/seg	**ωn**	2028.086815	rad/seg
ω	596.9026042	rad/seg	**ω**	596.9026042	rad/seg
Ω	1.133094089		**Ω**	0.294318073	

TABLA NO. 5. PERFIL CIRCULAR DE 1/2 PULG DE DIÁMETRO Y 1/16 PULG DE ESPESOR

Flexión			**Torsión**		
I	8.72937E-10	mts^4	**Ip**	1.74587E-09	mts^4
k	45841.97588	N/m	**Kt**	293.525042	Nm
ωn	677.0670268	rad/seg	**ωn**	2606.637903	rad/seg
ω	596.9026042	rad/seg	**ω**	596.9026042	rad/seg
Ω	0.881600463		**Ω**	0.228993296	

TABLA NO. 6. PERFIL CIRCULAR DE 3/4 PULG DE DIÁMETRO Y 1/32 PULG DE ESPESOR

Flexión			**Torsión**		
I	1.9002E-09	mts^4	**Ip**	3.80039E-09	mts^4
k	99788.15821	N/m	**Kt**	638.9411183	Nm
ωn	998.9402295	rad/seg	**ωn**	3845.816385	rad/seg
ω	596.9026042	rad/seg	**ω**	596.9026042	rad/seg
Ω	0.597535855		**Ω**	0.155208295	

3. DISCUSIÓN. El primer paso para diseñar la estructura es considerar un efecto crítico de la aplicación, se trata de las vibraciones mecánicas. Se considera este efecto principalmente ya que se desea evitar que la frecuencia natural de la estructura estuviese cercana a la frecuencia natural de operación. El efecto de estar operando a altas velocidades angulares hace que el desbalance que existe en las hélices aumente bastante y como el motor estará colocado en el extremo de una viga empotrada, esta funcionaría como un resorte cuya vibración podría resultar excesiva. El proceso de diseño se basa en dos resultados deseables: que el rango de frecuencias angulares utilizadas por el motor, se encuentre por debajo de la frecuencia angular de resonancia y que se optimice las dimensiones del perfil para obtener el menor peso. Debido a que se desconoce la magnitud de la fuerza de desbalance, no será posible conocer la magnitud de la vibración, sin embargo, el criterio de diseño empleado será utilizar un valor de relación de frecuencias angulares $\Omega<0.7071$, ya que considerando un sistema sin amortiguamiento, la función Hdesb es menor a 1 en esta condición. Esto implica que si el valor Me es pequeño será aún más pequeño si esta condición ocurre. Arriba de este valor podría comenzar a incrementarse la vibración a valores indeseables, y aún más, si Ω fuera muy cercano a 1, el sistema colapsaría. De los tres perfiles presentados en las Tablas 4 a la 6, se pude observar que Ω cumple la condición de ser menor a 0.7071 en torsión y flexión, para el perfil de aluminio circular hueco con diámetro de ¾" y espesor de 1/32".

D. DISEÑO CONTRA FRACTURA Y DEFLEXIONES.

1. DISEÑO EXPERIMENTAL.

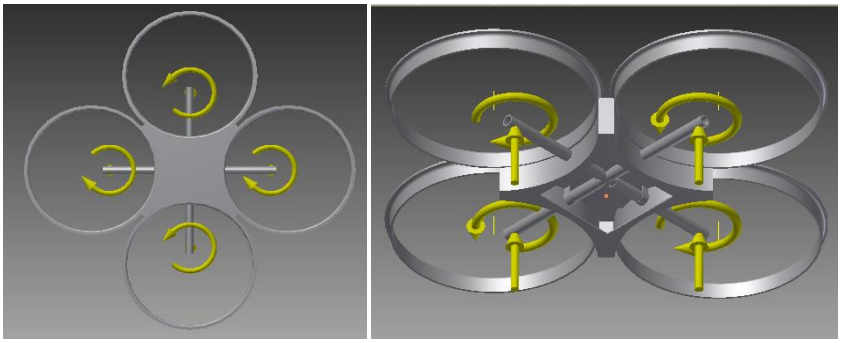
Posterior al modelado en tres dimensiones, se procede a realizar un análisis de las fuerzas externas del sistema, diseño contra deflexión y diseño contra la fractura. El análisis de fuerzas no requirió mayor procedimiento más que el análisis físico básico de sumatoria de fuerzas y sumatoria de momentos del sistema, ahora para el diseño contra deflexión y fractura se utilizó el software de Autodesk Inventor que por medio del método de elementos finitos obtiene el esfuerzo crítico y la deflexión máxima para el sistema. El esfuerzo se compara con el esfuerzo de fluencia del material para verificar si el sistema sufriría ruptura y posteriormente se compara que la deflexión no sea superior a 1 milímetro.

2. RESULTADOS

TABLA NO. 7. DATOS INICIALES PARA CÁLCULO DE ESFUERZOS Y DEFLEXIONES

	Aluminio	**Fibra de vidrio**
Módulo de Young (Gpa)	71.7	70
Densidad (Kg/m^3)	2710	2600
Resistencia a la fluencia (Mpa)	70	-
Resistencia ultima a la tensión (Mpa)	179	2410

FIGURA NO. 79. MODELADO EN INVENTOR PARA CÁLCULO DE ESFUERZO DE VON MISES Y DEFLEXIONES



Dado que la fibra de vidrio es un material frágil, no presenta valor de resistencia a la fluencia y su punto de comparación será la resistencia última a la tensión. Los resultados presentados por el software se muestran en las Figuras No. 80 y 81.

FIGURA NO. 80. GRADIENTE TRIDIMENSIONAL DE ESFUERZOS DE VON MISES

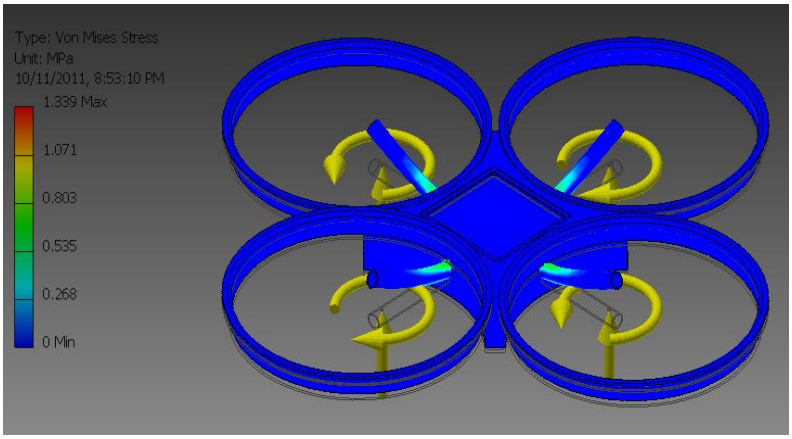
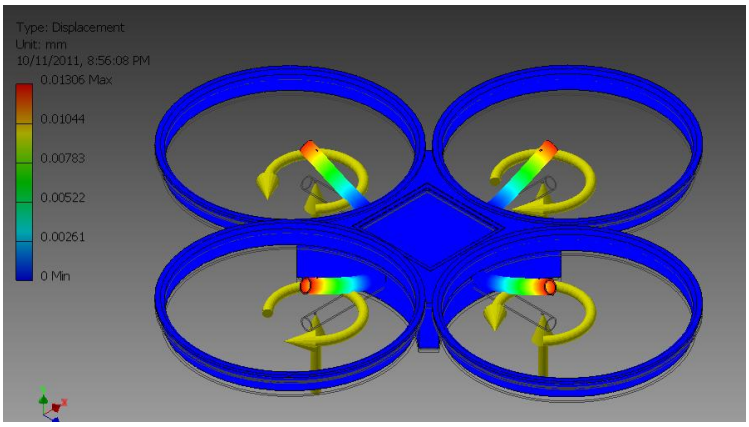


FIGURA NO. 81. GRADIENTE TRIDIMENSIONAL DE DEFLEXIÓN



3. DISCUSIÓN. Como era de esperarse, los esfuerzos no son muy grandes. El esfuerzo máximo de Von Mises presenta un valor de 1.339 MPa y se encuentra en la sección donde las vigas están empotradas dentro de la caja central, para ambos materiales existe gran seguridad en cuanto a la fluencia y fractura. Para el aluminio con una resistencia a la fluencia de 70Mpa se obtiene un factor de seguridad de 58.27 y para la fibra de vidrio con una resistencia última a la tensión de 2410, se obtiene un factor de seguridad de 1799.85. Estos valores tan altos de factores de seguridad se deben a que el perfil utilizado para soportar satisfactoriamente las vibraciones, es muy

grande para las pequeñas fuerzas que se ejercen, sin embargo, este perfil es el utilizado para que el sistema funcione correctamente.

En cuanto a las deflexiones, se observa en la Figura No. 81 que son menores a 1 milímetro, lo cual es considerado satisfactorio porque no es una deflexión que cause una fuerza horizontal significativa que pudiese perturbar al sistema.

E. CONSTRUCCIÓN DEL CUADCÓPTERO

1. DISEÑO EXPERIMENTAL. La siguiente etapa consistió en la implementación de la estructura; para ello se realizó un trabajo con fibra de vidrio y tubos de aluminio. Se realizó una pequeña cruz central de fibra de vidrio a donde se ensamblaron cuatro tubos de aluminio y luego se procedió a realizar los moldes que servirían como base en las demás partes del helicóptero, por ejemplo, las bancadas donde se colocarían los motores, la cavidad central para los componentes electrónicos y los parachoques. Los moldes fueron fabricados manualmente con cartón, madera y fibra de vidrio. Finalmente se pintó con aerosol.

La primera etapa en la creación de piezas en fibra de vidrio es crear un molde. Se pueden crear de diversos materiales como madera, aluminio, cartón y hasta de fibra de vidrio. Los moldes fueron realizados principalmente de madera y cartón, por su facilidad de trabajo. Estos moldes deben ser recubiertos de una grasa o un anti aglutinante para evitar que la resina se adhiera al molde, volviendo la pieza y/o el molde inservibles. Usualmente estos deben dejarse secando unos minutos para su mejor desempeño.

Para algunos moldes es necesario crear un molde en negativo. De esta manera se puede crear un molde en fibra de vidrio para la creación de varias piezas iguales. Estos moldes fueron utilizados para los protectores de las hélices y para las bancadas que soportan los motores. La estructura principal fue realizada con un molde de cartón para obtener la figura en forma de estrella y con planchas de aluminio, que tienen poca adherencia a la resina.

Luego de haber finalizado los moldes procedemos a crear la pieza. Se aplican capas de fibra de vidrio de manera longitudinal al molde. Se pueden agregar más capas de fibra de vidrio, dependiendo de la resistencia de la pieza, pero hay que tomar en cuenta que la pieza sería más ancha y de mayor masa. Ahora mezclamos la resina que vayamos a utilizar con el catalizador. No se debe de mezclar toda la resina con el catalizador, debido a que este se endurece rápidamente y desperdiciaríamos el excedente de este material.

Rápidamente aplicamos la resina al molde con fibra de vidrio y lo dejamos secando. Para aplicar la resina puede ser necesario un pincel o un rodillo para eliminar el aire acumulado. El tiempo de secado varía según la cantidad de catalizador utilizado y la exposición a la luz solar y condiciones ambientales. No es aconsejable utilizar demasiado catalizador porque este crearía un acabado menos fino.

La pieza puede terminar con algunos defectos, por lo que se suele utilizar masilla y lija pare mejorar su acabado. La función de la masilla es principalmente estética, proporcionando la mayoría de sus propiedades la fibra de vidrio y la resina. Para terminar se procede a pintar las piezas.

2. RESULTADOS

- Se diseñó la estructura con las dimensiones mostradas en la Figura No. 82, Figura No. 83 y Figura No. 84. (mm):

FIGURA NO. 82. VISTA SUPERIOR DEL DISEÑO DE LA ESTRUCTURA

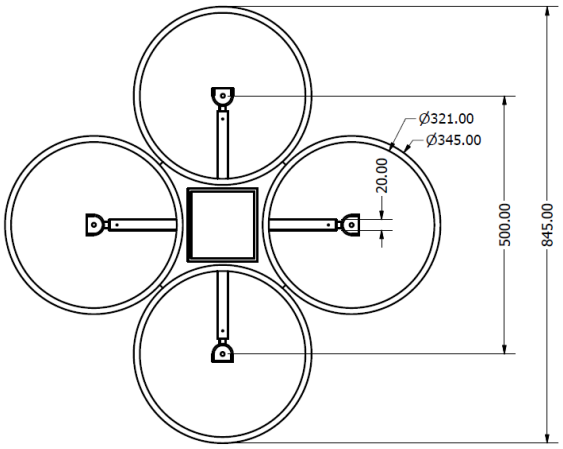


FIGURA NO. 83. VISTA LATERAL DEL DISEÑO DE LA ESTRUCTURA

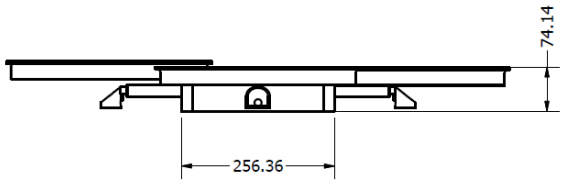


FIGURA NO. 84. VISTA ISOMÉTRICA DEL DISEÑO DE LA ESTRUCTURA

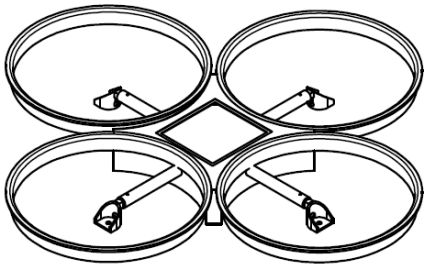


FIGURA NO. 85. MODELO DE LOS PROTECTORES DE LAS HÉLICES

FIGURA NO. 86. MOLDE COMPLETO DE LOS PROTECTORES

FIGURA NO. 87. PROTECTOR EN FIBRA DE VIDRIO

FIGURA NO. 88. CAJA CENTRAL DEL CUADCÓPTERO

FIGURA NO. 89. BANCADAS PARA LOS MOTORES

- Resultado final

FIGURA NO. 90. ESTRUCTURA PRINCIPAL DEL CUADCÓPTERO EN FIBRA DE VIDRIO

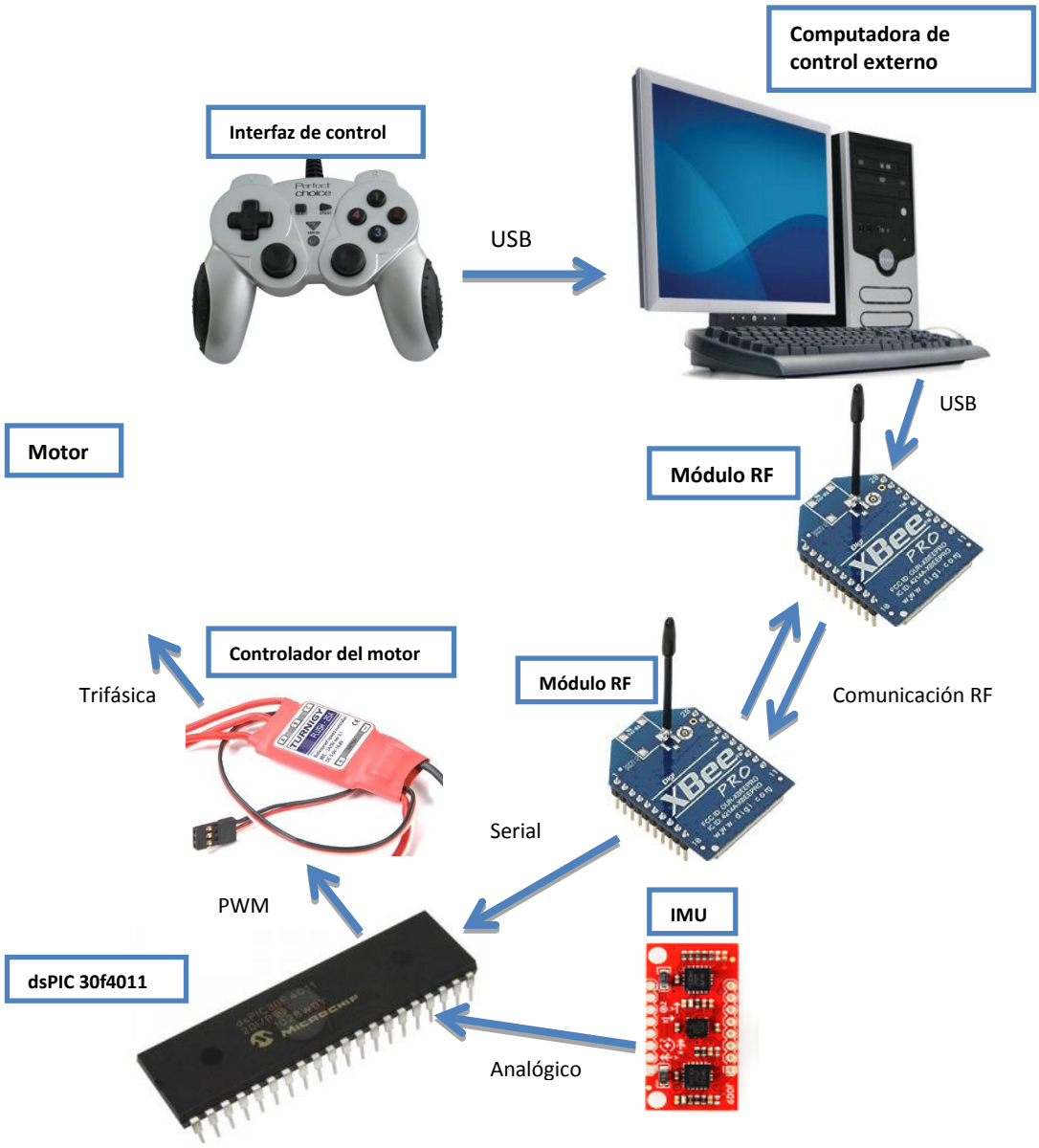
3. DISCUSIÓN. Su creación requirió de poco equipo y permitió bastante flexibilidad en el diseño. La estructura realizada es rígida y tiene un bajo peso. Permite el uso para el soporte de los circuitos de control del mismo, los motores y sus controladores. Los protectores reducen los daños a las hélices y motores.

VIII. ESTABILIZACIÓN Y CONTROL

A. BLOQUES DEL SISTEMA IMPLEMENTADO

1. DISEÑO EXPERIMENTAL. El sistema del cuadcóptero se presenta en la Figura No. 91.

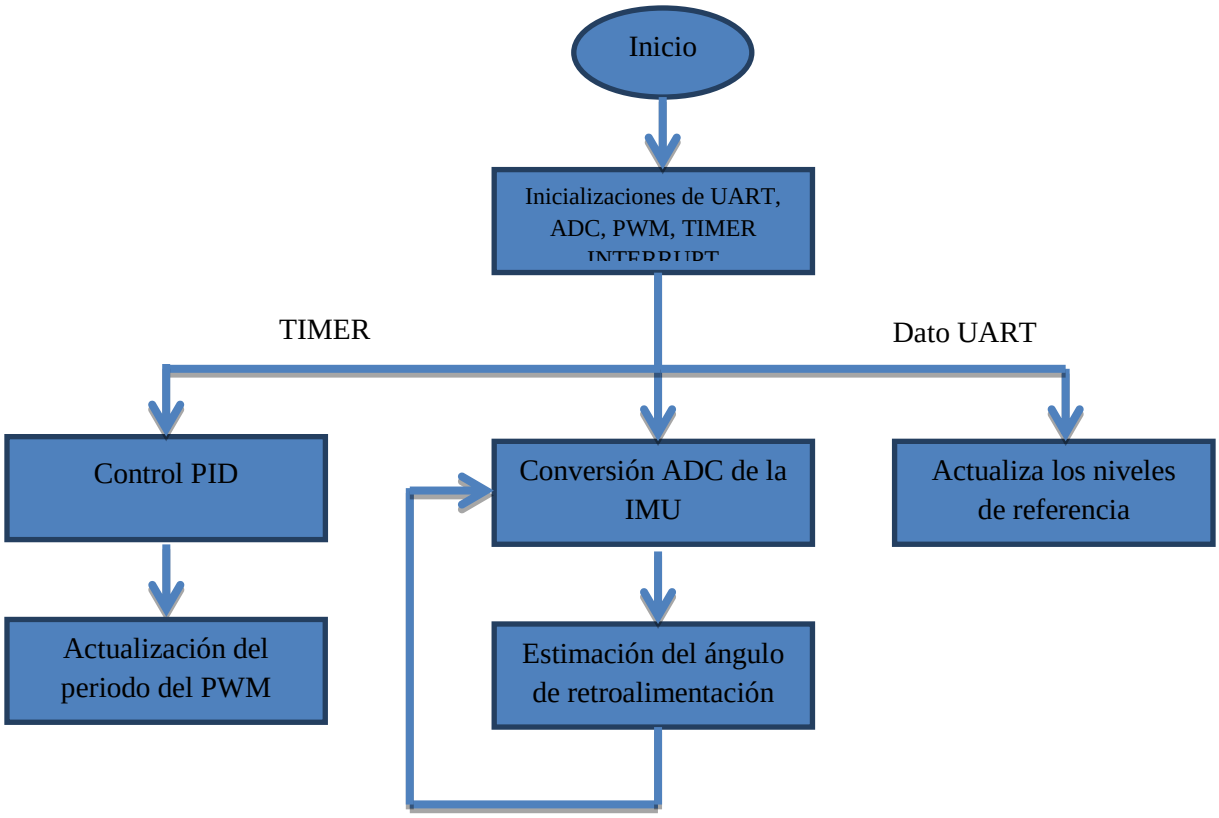
FIGURA NO. 91. SISTEMA ELECTRÓNICO DEL CUADCÓPTERO



2. RESULTADOS

- Diagrama de flujo del programa del dsPIC se encuentra en la Figura No. 92.

FIGURA NO. 92. DIAGRAMA DE FLUJO DE CONTROL INTERNO DEL CUADCÓPTERO



3. DISCUSIÓN. Se utilizó el dsPIC30F4011 por su velocidad de conversión ADC, 16 bits de procesamiento, y sus 5 salidas de PWM. Se implementó el control PID a intervalos definidos, y sus niveles de referencia se actualizan desde un módulo RF.

B. MODELO MATEMÁTICO Y CARACTERIZACIÓN DEL SISTEMA

1. DISEÑO EXPERIMENTAL. En esta etapa se procedió a realizar las experimentaciones necesarias y lograr determinar las funciones de transferencia entre la señal de ingreso de los controladores de los motores (PWM) y su efecto en la inclinación, la elevación, la rotación y el desplazamiento horizontal. Para cumplir con dicho requerimiento se realizaron básicamente dos pruebas: la primera, consistió en obtener la velocidad y fuerza a diversas señales de entrada, tomando en cuenta también la forma transitoria en que un valor de fuerza en particular llega a establecerse (conducta transitoria). Para medir la velocidad se utilizó un optoacoplador que generaba un pulso cada vez que una parte de la hélice lo interrumpía, estos pulsos eran leídos por el osciloscopio en donde se observó una onda cuadrada cuya frecuencia proporcionaba la información de velocidad angular. En el caso de la fuerza se utilizaron sensores de fuerza, se colocó una plancha de madera sostenida por dichos sensores y el helicóptero se colocó en un banco más alto de donde se encontraba la plancha, de manera que la hélice quedó arriba y al girar la fuerza de empuje fue leída directamente por sensores de fuerza que eran jalados por la plancha. Finalmente las mediciones se procesaron en un computador para obtener los resultados requeridos de sustentación y velocidad.

La segunda prueba realizada fue obtener el torque del motor dado una señal de PWM, para realizar esta prueba se parte del hecho que la mayoría de la potencia mecánica cuando ya se ha establecido una velocidad de giro, sirve solamente para vencer la fricción y el arrastre que es precisamente el torque del motor. De esta manera, midiendo la potencia eléctrica por la eficiencia del motor y la velocidad angular a cierto PWM se determina la función de transferencia entre el torque y PWM. La velocidad se midió, al igual que en la prueba anterior, con un optoacoplador y un osciloscopio. La potencia eléctrica se obtuvo midiendo el voltaje y la corriente a cierto PWM.

2. CÁLCULOS Y RESULTADOS. En esta sección se realizará un modelo teórico que describa a los seis ejes de libertad del sistema en función de las entradas PWM en los controladores de los motores, es decir, el modelo del cuadcóptero en forma de lazo abierto. De esta manera quién se encargue del control será capaz de retroalimentar el grado de libertad deseado para organizar las señales PWM de tal forma que produzcan una respuesta estable y controlada.

En general existen tres funciones de transferencia iniciales para cada eje, que van desde la señal PWM hasta la fuerza o torque que produce cada motor. Dichas funciones de transferencia serán las correspondientes al controlador del motor, al motor y a la hélice para producir un torque y una fuerza. La señal PWM del controlador se traduce en potencia eléctrica, la potencia eléctrica se traduce en velocidad angular y la velocidad angular se traduce en torque y en sustentación. El problema que presentan estas funciones de transferencia es que de las tres, dos no son lineales, esto significa que el modelado por medio de la transformada de Laplace no sería útil y terminaría siendo complicado modelarlo. Para evitar este incidente y como aproximación al modelo real del cuadcóptero nos basamos en dos hechos, el primero es que el controlador permite obtener una función lineal entre PWM y fuerza o entre PWM y torque. El otro hecho es que podemos estimar la función de transferencia de un sistema como una amplificación y un retardo, de la siguiente manera.

FIGURA NO. 93. FUNCIÓN DE TRANSFERENCIA

$$H = \frac{A}{s + \tau}$$

Donde para nuestros propósitos esta aproximación predice muy bien lo que ocurre en el sistema debido que el retardo es principalmente por el conjunto motor más hélice, cuyo retardo está en función de la inercia que debe acelerar.

De esta manera obtendríamos las siguientes expresiones para la transferencia entre la señal PWM y Fuerza de la siguiente manera.

FIGURA NO. 94. TRANSFERENCIA ENTRE LA SEÑAL PWM Y FUERZA

$$\frac{\mathrm{Fn}}{\mathrm{pwm.\,n}} = \frac{\mathrm{An}}{\mathrm{s} + \tau\mathrm{n}}$$

Donde n representa el índice correspondiente para cada motor, F la fuerza de sustentación, pwm la señal colocada en la entrada, A la amplificación o atenuación y τ la constante de tiempo para la respuesta del motor. De la misma manera para el torque producido por cada motor se tiene la siguiente expresión

FIGURA NO. 95. TORQUE PRODUCIDO POR MOTOR

$$\frac{\mathrm{Tn}}{\mathrm{pwm.\,n}} = \frac{\mathrm{Bn}}{\mathrm{s} + \tau\mathrm{n}}$$

Donde T representa el torque y B la atenuación o amplificación. El PWM y el τ son iguales, ya que se trata de la misma señal que entra a los motores y del mismo tiempo en establecerse una salida.

a. Modelo para la inclinación. El mismo modelo utilizado para la inclinación sobre el eje X se utiliza para la rotación sobre el eje Y. Para ello se utiliza la sumatoria de momentos sobre el eje X

FIGURA NO. 96. SUMATORIA DE MOMENTOS SOBRE EJE X

$$\Sigma \mathrm{Tx} = \mathrm{Ixx} * \frac{\mathrm{d}^2\vartheta}{\mathrm{dt}^2}$$

Dado que la transformada de Laplace de la suma es la suma de las transformadas, al aplicar la transformación a la ecuación anterior se obtiene la siguiente expresión en el dominio de Laplace.

FIGURA NO. 97. ÁNGULO DE INCLINACIÓN DOMINIO DE LAPLACE

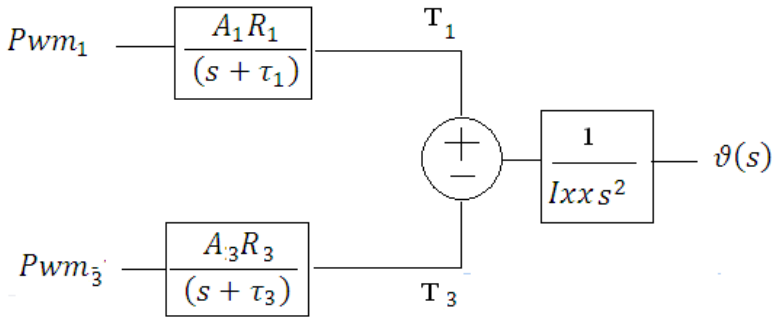
$$\vartheta(\mathrm{s}) = \frac{\Sigma \mathrm{Tx}\,(\mathrm{s})}{\mathrm{Ixx} * \mathrm{s}^2}$$

Por lo tanto, el ángulo de inclinación depende únicamente del torque producido por las fuerzas de los dos motores y sus respectivos radios hacia el centro de masa, los cuales son contrarios. La función de inclinación se expresa en términos de las dos señales PWM de la siguiente manera.

FIGURA NO. 98. FUNCIÓN DE INCLINACIÓN EN TÉRMINOS DE DOS SEÑALES PWM

$$\vartheta(s) = \frac{1}{Ixx\, s^2} * \left(\frac{A_1 R_1 Pwm_1}{(s+\tau_1)} - \frac{A_3 R_3 Pwm_3}{(s+\tau_3)}\right)$$

FIGURA NO. 99. DIAGRAMA DE BLOQUES PARA LA INCLINACIÓN DE UN EJE EN SISTEMA DE LAZO ABIERTO



b. Modelo para rotación sobre el eje z. Para la rotación en el eje z se deberá tomar en cuenta el torque ocasionado por los cuatro motores, y se deberá cumplir con sumatoria de momentos alrededor del eje z.

FIGURA NO. 100. SUMATORIA DE MOMENTOS ALREDEDOR DE EJE Z

$$\Sigma Tz = Izz * \frac{d^2\Psi}{dt^2}$$

Dado que la transformada de Laplace de la sumatoria es la sumatoria de las transformadas, es válida la siguiente expresión.

FIGURA NO. 101. ROTACIÓN EN DOMINIO LAPLACE

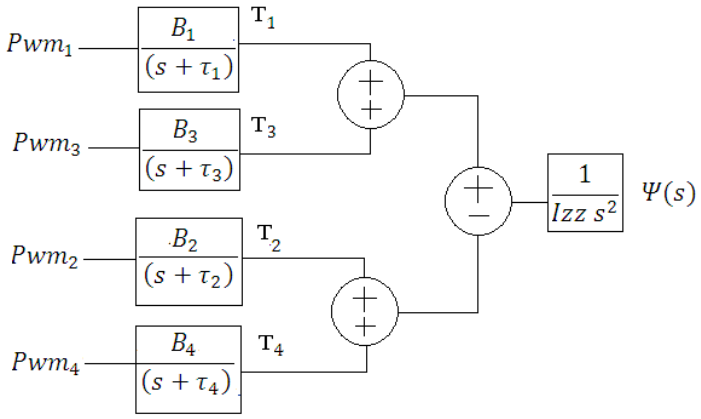
$$\Psi(s) = \frac{\Sigma Tz\,(s)}{Izz * s^2}$$

Ahora bien, dado que el motor 1 y 3 giran en un sentido y el motor 2 y 4 giran en otro sentido, la función de rotación en término de los 4 PWM se puede escribir de la siguiente forma.

FIGURA NO. 102. FUNCIÓN DE ROTACIÓN EN TÉRMINO DE LOS 4 PWM

$$\Psi(s) = \frac{1}{Izz\, s^2} * \left(\frac{B_1 Pwm_1}{(s+\tau_1)} + \frac{B_3 Pwm_3}{(s+\tau_3)} - \frac{B_2 Pwm_2}{(s+\tau_2)} - \frac{B_4 Pwm_4}{(s+\tau_4)}\right)$$

FIGURA NO. 103. DIAGRAMA DE BLOQUES PARA ROTACIÓN SOBRE EJE Z EN SISTEMA DE LAZO ABIERTO



c. Modelo para sustentación y elevación. Para la elevación sobre el eje z se debe cumplir con la siguiente expresión.

FIGURA NO. 104. SUMATORIA DE FUERZAS EN EJE Z

$$\Sigma Fz = m * \frac{d^2 z}{dt^2}$$

O bien como ya se ha explicado con anterioridad

FIGURA NO. 105. ELEVACIÓN EN DOMINIO LAPLACE

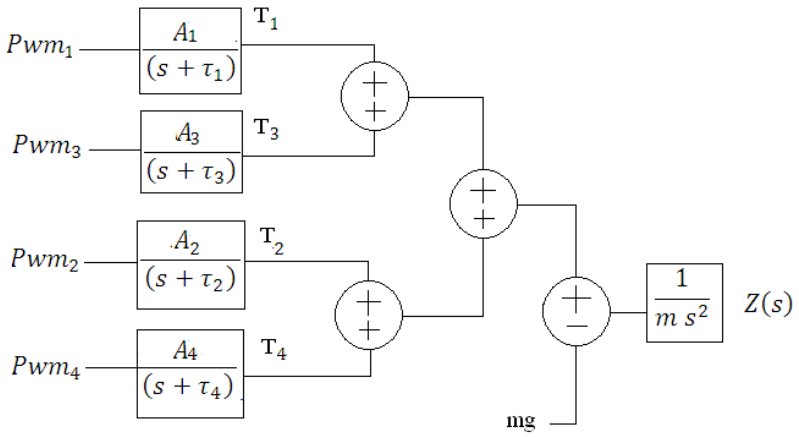
$$Z(s) = \frac{\Sigma Fz\,(s)}{m * s^2}$$

La sumatoria de fuerzas sobre el eje z está compuesta por la sustentación de los cuatro motores y el peso. Su ecuación se puede expresar de la siguiente manera.

FIGURA NO. 106. FUNCIÓN DE ELEVACIÓN

$$Z(s) = \frac{1}{m\,s^2} * \left(\frac{A_1 Pwm_1}{(s+\tau_1)} + \frac{A_2 Pwm_2}{(s+\tau_2)} + \frac{A_3 Pwm_3}{(s+\tau_3)} + \frac{A_4 Pwm_4}{(s+\tau_4)} - mg\right)$$

FIGURA NO. 107. DIAGRAMA DE BLOQUES PARA LA ELEVACIÓN EN SISTEMA DE LAZO ABIERTO



d. Modelo para la traslación horizontal. Para el avance en X se utiliza el mismo modelo que para el avance en Y. Para que ocurra movimiento horizontal, debe ocurrir cierta inclinación para producir una fuerza resultante en el eje esperado, y además se debe cumplir la siguiente relación en el caso de desplazamiento sobre el eje X.

FIGURA NO. 108. SUMATORIA DE FUERZAS EN EJE X

$$\Sigma \mathrm{Fx} = \mathrm{m} * \frac{\mathrm{d}^2\mathrm{x}}{\mathrm{dt}^2}$$

O expresado en el dominio de Laplace

FIGURA NO. 109. TRASLACIÓN EN DOMINIO DE LAPLACE

$$\mathrm{X(s)} = \frac{\Sigma \mathrm{Fx\,(s)}}{\mathrm{m} * \mathrm{s}^2}$$

La fuerza ocasionada para el desplazamiento horizontal será

FIGURA NO. 110. FUERZA PARA DESPLAZAMIENTO HORIZONTAL

$$\mathrm{Fx} = \mathrm{Fsustentación} * \mathrm{sen}(\Phi)$$

Esta función ocasiona grandes problemas porque si estamos multiplicando dos funciones en tiempo se debería hacer la convolución en el dominio de Laplace y además el argumento del seno hace no lineal la función. Sin embargo, para un nivel de elevación en especial, se tendría una sustentación aproximadamente constante y para ángulos pequeños la aproximación sen(x)=x es válida. Para que un cuadcóptero vuele

establemente no deberá tener ángulos mayores de 45 grados, región donde la aproximación tiene un máximo de 11% de error, lo cual se considerará aceptable. De esta cuenta, la fuerza en x será.

FIGURA NO. 111. FUERZA PARA DESPLAZAMIENTO HORIZONTAL SIMPLIFICADA

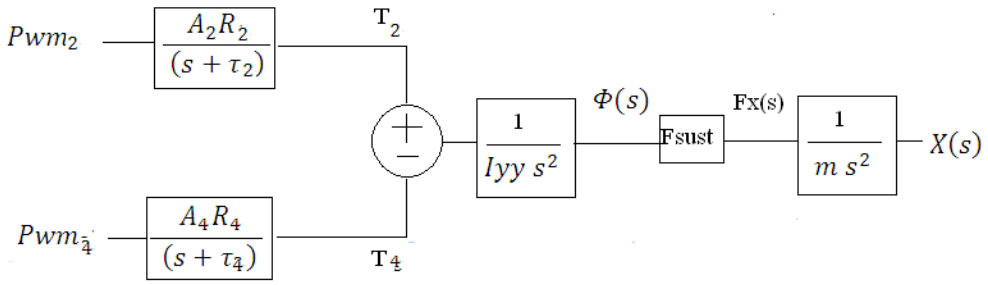
$$Fx = Fsust * \Phi$$

Y la función de desplazamiento horizontal se puede aproximar mediante la siguiente expresión, haciendo despreciable la fricción por resistencia del aire.

FIGURA NO. 112. FUNCIÓN DE DESPLAZAMIENTO HORIZONTAL

$$X(s) = \frac{1}{m\,s^2} * \left(\frac{Fsust}{Iyy\,s^2} * \left(\frac{A_2 R_2 Pwm_2}{(s+\tau_2)} - \frac{A_4 R_4 Pwm_4}{(s+\tau_4)} \right) \right)$$

FIGURA NO. 113. DIAGRAMA DE BLOQUES PARA EL AVANCE HORIZONTAL EN SISTEMA DE LAZO ABIERTO



Ya que se posee un diagrama de bloques de las funciones de transferencia que describen los movimientos básicos del cuadcóptero, se procede a caracterizar ciertos bloques para tener constantes específicas. La primera etapa de la caracterización del sistema es determinar la función de transferencia PWM versus sustentación y PWM versus torque, para ello se toman datos de cuánta fuerza se genera y cuánto torque se genera al colocar una señal de PWM especificada.

FIGURA NO. 114. RESPUESTA DE FUERZA VERSUS TIEMPO

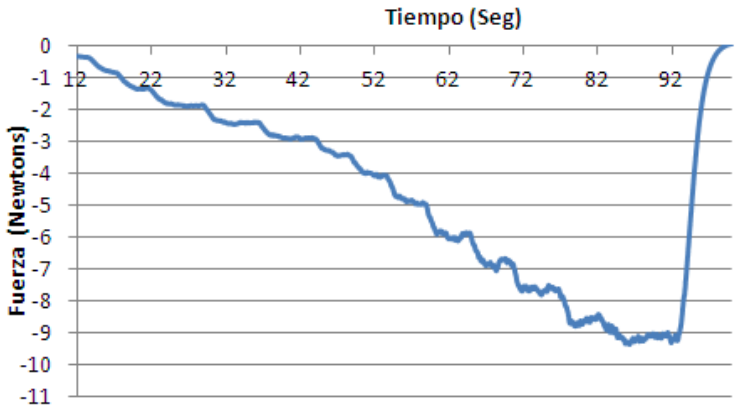
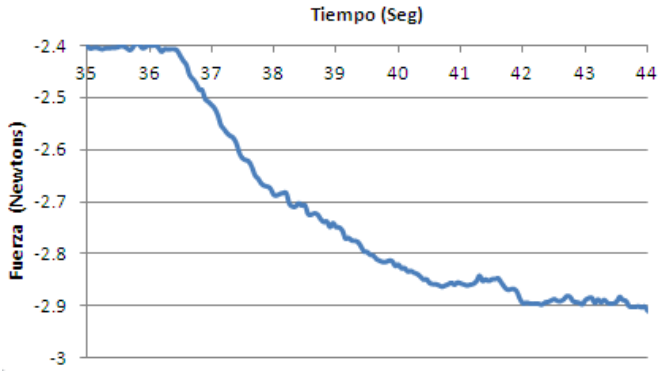


FIGURA NO. 115. RESPUESTA DE FUERZA AL ESCALÓN ANTE UNA SEÑAL DE ENTRADA PWM



La gráfica de respuesta de fuerza al escalón se utiliza para determinar la constante de tiempo (τ), que será tomada en cuenta en los modelos matemáticos del sistema. La fuerza a la que tiende esta gráfica es a 2.9N, partiendo de 2.4 N esto quiere decir que sube 0.5N. Debido a que una función de transferencia es de la forma.

FIGURA NO. 116. DETERMINACIÓN DE CONSTANTE DE TIEMPO

$$H = \frac{A}{s + \tau}$$

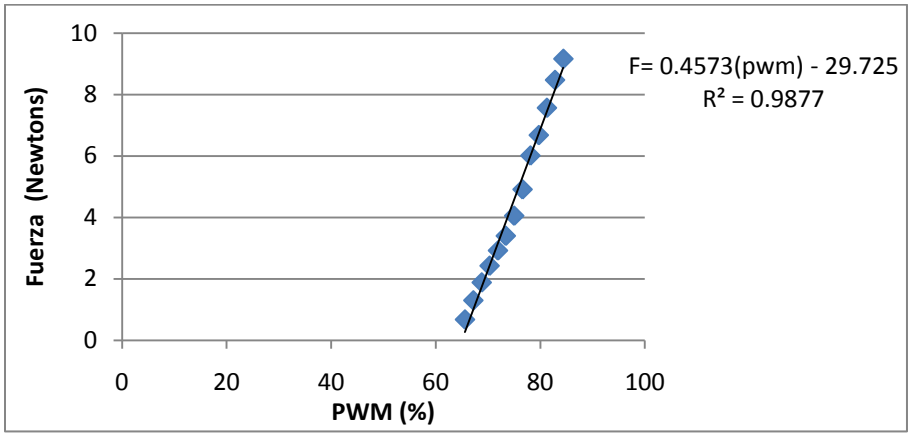
Tiene una respuesta al escalón de la siguiente forma.

$$F = K - Ke^{-\tau t}$$

Donde A en nuestro caso es 0.5N. Por lo tanto de la gráfica podemos conocer cuánto tiempo tarda en llegar a 0.6K = 0.3 desde 2.4. Esto será

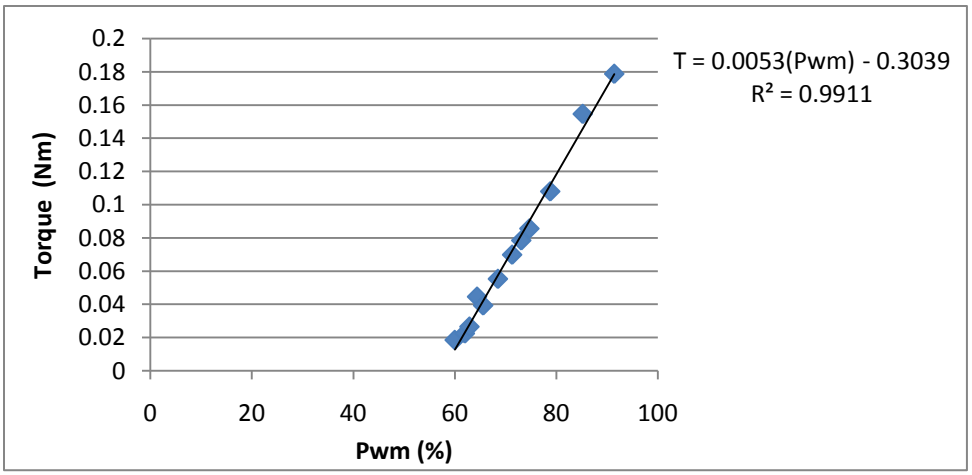
$$\tau = -\frac{\ln(0.4)}{2} = 0.4581$$

FIGURA NO. 117. RELACIÓN ENTRE LA SEÑAL PWM Y FUERZA



De la gráfica PWM versus fuerza se obtiene la constante de proporcionalidad A para la función de transferencia de fuerza. Para el sistema de control se deberá asumir que el nuevo cero de PWM es desde que inicia a operar (cerca del 65% real), de manera que la gráfica este centrada en cero, solamente por conveniencia y simplificación. Por lo tanto A=0.4573* τ = 0.2095.

FIGURA NO. 118. RELACIÓN ENTRE LA SEÑAL PWM Y TORQUE



De la gráfica PWM versus torque se obtiene la constante de proporcionalidad B para la función de transferencia de torque. Al igual que para la función de transferencia

de fuerza se deberá asumir que el nuevo cero de PWM es desde que inicia a operar (cerca del 65% real). Por lo tanto B= 0.0053* τ= 0.002428

Con los datos anteriores las funciones de transferencia ente PWM versus fuerza y PWM versus torque quedaría de la siguiente manera.

$$\frac{\text{Fuerza}}{\text{PWM}} = \frac{0.2095}{(s + 0.4581)}$$

$$\frac{\text{Torque}}{\text{PWM}} = \frac{0.002428}{(s + 0.4581)}$$

FIGURA NO. 119. RELACIÓN ENTRE LA SEÑAL VELOCIDAD ANGULAR Y FUERZA

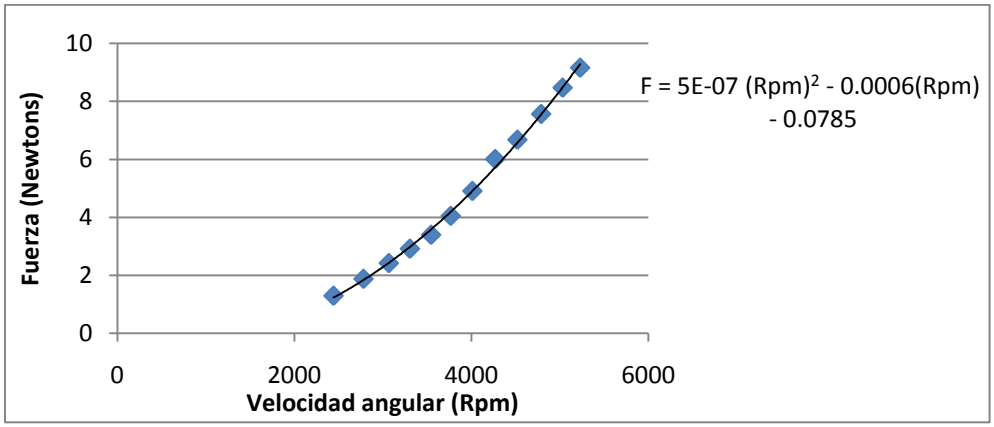


FIGURA NO. 120. RELACIÓN ENTRE VELOCIDAD ANGULAR Y TORQUE

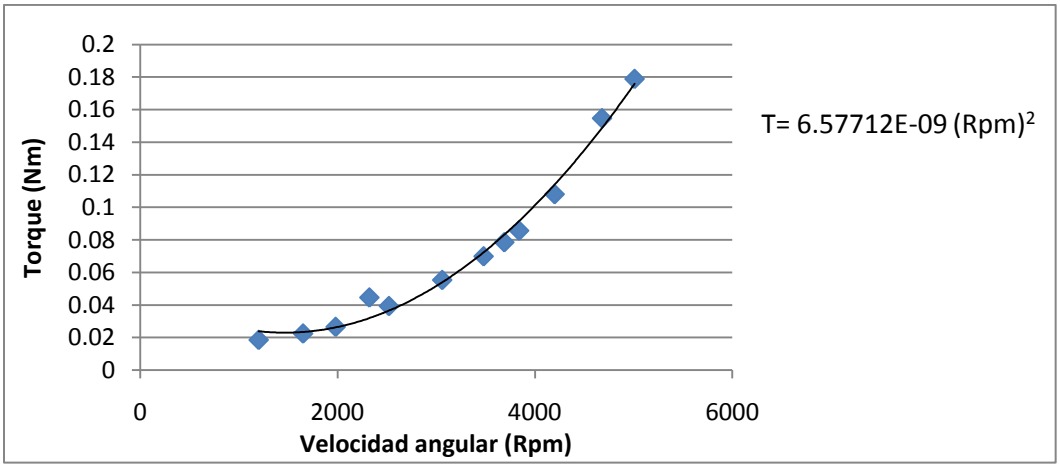
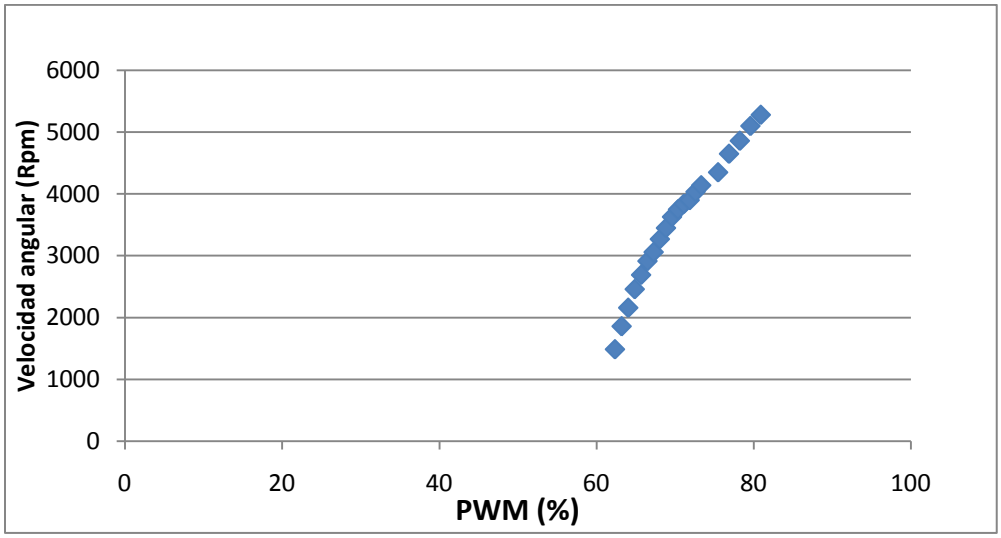


FIGURA NO. 121. RELACIÓN ENTRE LA SEÑAL PWM Y VELOCIDAD ANGULAR



3. DISCUSIÓN. Para la función de fuerza versus velocidad angular, se observa que la función es cuadrática tal y como lo predice la teoría, sin embargo, presenta, cierto desplazamiento debido a que el sistema comienza a ejercer sustentación significativa hasta ciertas revoluciones por minuto. Para 5700 rpm la regresión cuadrática ofrece un valor de fuerza de 12.7465 N y según la curva característica de la hélice se determinó que para 5700 rpm se obtiene un valor de fuerza de 11N, esto representa un 15.89% de error.

Para la función de torque versus velocidad angular, se observa que cumple con ser cuadrática tal y como lo describe la teoría. Esta gráfica se encuentra centrada en cero. Para 5700 rpm la regresión cuadrática ofrece un valor de torque de 0.21369 Nm y según la curva característica de la hélice se determinó que para 5700 rpm se obtiene un valor de torque de 0.2 Nm, esto representa un 6.84% de error.

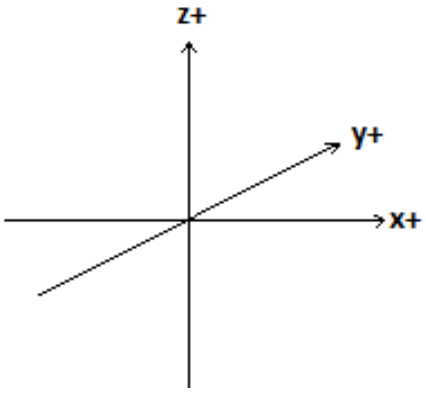
La curva experimental de velocidad angular versus PWM, se observa que no es lineal, de hecho tiene una curva un poco particular. Esto es debido a que el controlador del motor configura la potencia de tal forma que permita generar una sustentación lineal con relación al PWM.

C. OBTENCIÓN DEL ÁNGULO DE INCLINACIÓN

1. DISEÑO EXPERIMENTAL. Para presentar el modelo de la obtención del ángulo de inclinación es necesario presentarlo en tres partes principales: la obtención del ángulo según el acelerómetro, según el giroscopio y la unión de ambas mediciones para obtener una mejor estimación del ángulo que las anteriores individualmente.

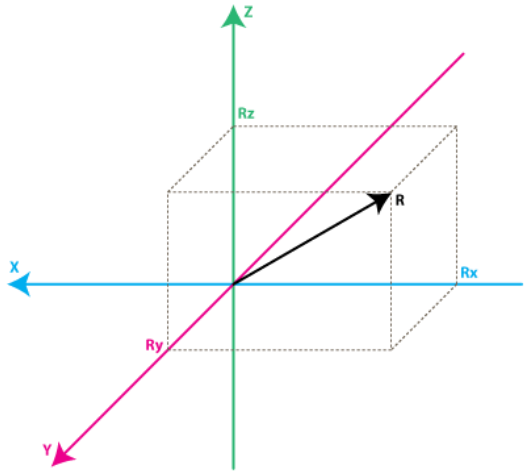
a. Obtención del ángulo con el acelerómetro. Se realizó un modelo para la obtención del ángulo, utilizando trigonometría. Se utilizaron los ejes de referencia mostrados en la Figura No. 122:

FIGURA NO. 122. EJES DE REFERENCIA



Se analizó el comportamiento del acelerómetro y se modelo para obtener el vector de gravedad proyectado en sus ejes. El modelo que se obtuvo del acelerómetro se muestra en la Figura No. 123.

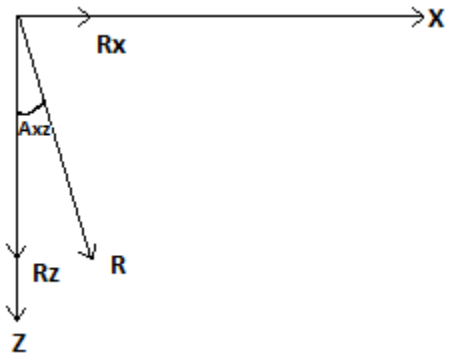
FIGURA NO. 123. MODELO DEL ACELERÓMETRO



Donde R es el vector de gravedad medido por el sensor en estado estático.

Proyectado en un plano el problema se limita a la Figura No. 124.

FIGURA NO. 124. ACELERACIÓN PROYECTADA EN EL PLANO XZ



Donde:

- Axz es el ángulo del vector de aceleración proyectado en el plano XZ.
- Rx es la aceleración sobre el eje X del sensor
- Rz es la aceleración sobre el eje Z del sensor

FIGURA NO. 125. DEDUCCIÓN E IMPLEMENTACIÓN DE ÁNGULO

Al aplicar trigonometría se obtiene:

$$\tan(Axz) = Rx/Rz$$

Resolviendo por el ángulo:

$$Axz = \arctan(\frac{Rx}{Rz})$$

De manera similar se obtiene el ángulo sobre el plano YZ:

$$Ayz = \arctan(\frac{Ry}{Rz})$$

Su implementación es la siguiente:

$$Axz[n] = atan2(Rx[n], Rz[n])$$

Se utilizó atan2 porque mapea el ángulo centrado en cero. Sobre el plano XY no se representa su ángulo de "Yaw", sino una proyección de la inclinación en los otros ejes. Debido a esto no se puede obtener una aproximación del ángulo utilizando el acelerómetro, por lo que no lo podemos controlar con esta información.

El problema con estas mediciones es que son válidas en estado estático, ya que el acelerómetro también mide las aceleraciones dinámicas del objeto. Pero aún en estado dinámico se tiene esta componente, por lo que al filtrarla se obtiene una mejor aproximación. Esto se debe a que es un sensor de aceleración, por lo que aun filtrando el ángulo, si se tiene otra aceleración constante o a frecuencias bajas, estas se representarían en la lectura.

Una mejor aproximación al ángulo, filtrando su señal sería:

FIGURA NO. 126. MEJOR APROXIMACIÓN AL ÁNGULO

$$Axz[n] = Axz[n-1] * \alpha + atan2(Rx[n], Rz[n]) * (1 - \alpha)$$

Donde α tiene un valor entre cero y uno y presenta una frecuencia de corte dependiente de la velocidad de muestreo. Hay que tomar en cuenta que el filtro presenta un desfase (retardo), y que hay que encontrar la frecuencia correcta para filtrar, donde se minimiza el ruido por aceleraciones distintas a la gravedad.

b. Obtención del ángulo CON el giroscopio. El giroscopio nos presenta datos de velocidad angular sobre los mismos tres ejes del acelerómetro. Para obtener el ángulo es necesario integrar la velocidad. La integración se lleva a cabo en el microcontrolador, por lo que es necesario modelar esta operación numéricamente.

Nuestro modelo del sensor es:

FIGURA NO. 127. MODELO DEL SENSOR

$$\emptyset(t) = \int \omega \, dt$$

Donde

- $\emptyset$ es el ángulo
- ω es la velocidad angular

FIGURA NO. 128. DEDUCCIÓN E IMPLEMENTACIÓN DE FUNCIÓN DE ÁNGULO EN GIROSCOPIO

Aproximando la integral utilizando rectángulos se obtiene:

$$\int \omega \, dt = lim_{n \to \infty} \sum_{i=1}^{n} \omega_i \, \Delta t$$

Si nuestra velocidad de muestreo es suficientemente rápida podemos aproximar el ángulo de la siguiente manera:

$$\emptyset_{nT} = \sum_{i=1}^{nT} \omega_i \, T$$

El cálculo en la ecuación de diferencias en tiempo real sería:

$$\emptyset[n] = \emptyset[n-1] + \omega[n] * T$$

La implementación puede ser mejorada utilizando integración por trapezoides y filtrando la señal de entrada.

Sobre los ejes de interés:

$$Axz[n] = Axz[n-1] + Wxz\,[n] * T$$

$$Ayz[n] = Ayz[n-1] + Wyz\,[n] * T$$

El problema con esta implementación es que tiene un desplazamiento con el tiempo ("*bias*"), dicho de otra manera, el nivel del ángulo cero varia con el tiempo a pesar que el sistema esté estático. Se podría implementar un filtro pasa alta para obtener una mejor aproximación del ángulo, pero perderíamos información sobre el verdadero ángulo.

c. Obtención conjunta del ángulo. Ambas maneras de medir tienen lados negativos y positivos. El ángulo del acelerómetro es muy ruidoso y puede medir aceleraciones externas, el ángulo según del giroscopio es bueno dinámicamente, pero tiene un deslizamiento de su señal a frecuencias bajas que limita su uso por lapsos pequeños.

Para unir ambas lecturas se utilizó un filtro complementario o ponderado. Lo que se hace es ponderar la confianza entre ambos sensores para obtener una mejor lectura conjuntamente. Adicionalmente el filtro posee propiedades de un filtro pasa baja y de filtro pasa alta, que son los problemas que presentan los sensores individualmente.

El filtro es el siguiente:

FIGURA NO. 129. MODELO DE FILTRO

$$z = \alpha * x + (1 - \alpha) * y$$

Donde:

- α es el nivel de confianza
- "x" y "y" son nuestras estimaciones
- "z" nuestra estimación conjunta

En esta implementación se tiene la limitante que α es un valor entre cero y uno.

En nuestra implementación se tiene:

FIGURA NO. 130. MODELO DE ÁNGULO COMBINADO

$$Axz = \alpha * AxzGiro + (1 - \alpha) * AxzAcc$$

De forma alternativa se puede plantear el filtro como una ponderación en conjunta, donde α solo está limitada a ser positiva, pero tiene los mismos efectos:

$$Axz = (\beta * AxzGiro + AxzAcc)/(1 + \beta)$$

Donde

- AxzGiro es la estimación del ángulo utilizando el giroscopio
- AxzAcc es la estimación del ángulo utilizando el acelerómetro.

Para evitar el sesgo se utiliza esta aproximación como valor anterior del ángulo al momento de integrar la velocidad angular. Se puede realizar una ponderación dinámica del ángulo, teniendo un parámetro de confianza variable. En nuestro caso este puede ser la magnitud de la aceleración medida por el acelerómetro. Esto es porque idealmente nosotros medimos la aceleración de la gravedad, pero si tenemos aceleraciones externas entonces este valor varía.

La calibración del filtro ponderado se llevó a cabo en tiempo real. Se modificaba su valor de confianza y se varía el ángulo del sensor. El movimiento se realizaba rápido y lento sobre sus 6 ejes de libertad hasta obtener una respuesta deseada.

Si se presenta una pendiente en el ángulo, entonces se disminuye el valor de confianza en el giroscopio. Si se presenta ruido en la señal, y variaciones del ángulo bruscas al momento de desplazarlo, entonces se aumenta la confianza en el giroscopio.

2. RESULTADOS

- La obtención del ángulo según el acelerómetro es de la siguiente manera:

$$Axz[n] = atan2(Rx[n], Rz[n])$$

- La obtención del ángulo según el giroscopio es de la siguiente manera:

$$Axz[n] = Axz[n-1] + Wxz\,[n] * T$$

- La obtención conjunta del ángulo es:

$$Axz = (\alpha * AxzGiro + AxzAcc)/(1+\alpha)$$

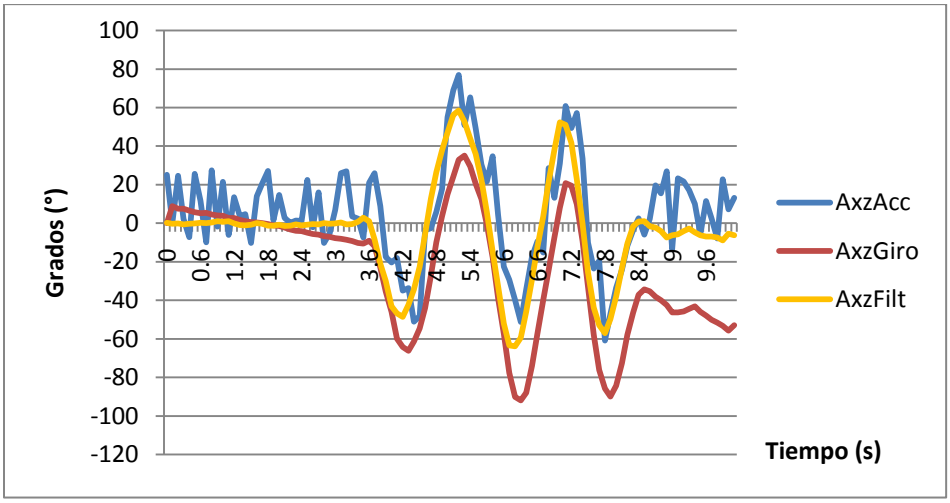
- El α utilizado es:

$$\alpha = 340 + ABS(2500 * errorAcc)$$

- Se tiene una oscilación en estado estable del sensor con una magnitud menor a un grado.

- En la Figura No. 131 se muestra la respuesta del ángulo utilizando los métodos anteriores.

FIGURA NO. 131. ÁNGULO FILTRADO VS ÁNGULO DEL ACELERÓMETRO Y DEL GIROSCOPIO



3. DISCUSIÓN. Se logró una estimación del ángulo obteniendo información conjunta del giroscopio y acelerómetro. Para la obtención del ángulo según el acelerómetro fue necesario aplicar trigonometría básica y para el giroscopio fue necesario integrarlo. Para obtener una estimación conjunta se implementó un filtro ponderado, para el cual se obtuvo una estimación inicial de su valor y luego se calibro manualmente.

El ángulo obtenido por el filtro fue mejor que cada estimación del ángulo independientemente. Se observó una pequeña oscilación del ángulo, esta puede afectar el comportamiento del control PID, pero se obtuvo una respuesta rápida y precisa para la aplicación.

Este acercamiento a la obtención del ángulo fue intuitivo y de implementación simple, por lo que se sugiere esta solución para estos tipos de sensores, como alternativa a un filtro Kalman. Este tipo de filtro puede ser utilizado en otras aplicaciones y otros sensores con características similares, como por ejemplo la obtención de la posición utilizando GPS (sistema de posicionamiento global) y un tacómetro.

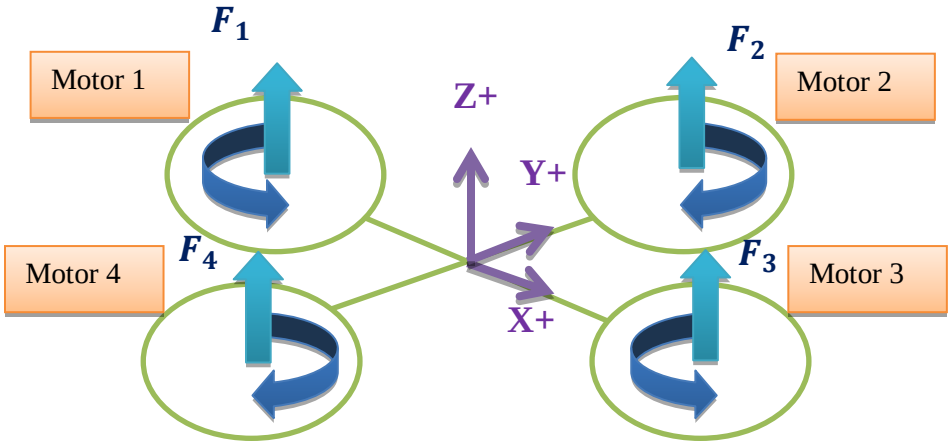
No siempre es necesario implementar el filtro dinámico, pero preferimos utilizarlo para evitar mayores interferencias. Se obtuvo una respuesta satisfactoria, por lo que no existió la necesidad de integrar utilizando trapezoides.

No se logra obtener información sobre el eje Z del ángulo utilizando el acelerómetro, por lo que no se logra obtener una lectura fiable del ángulo en este eje. Debido a esto se crea un control de rotación sobre este eje y no uno de ángulo. Esto se podría mejorar agregándole una brújula sobre este eje.

D. CONTROL DEL CUADCÓPTERO

1. ANÁLISIS DEL SISTEMA. Como se presentó con anterioridad, el sistema se puede modelar como en la Figura No. 132, tomando en cuenta que el modelo de los motores es cercano a lineal en la transferencia a fuerza:

FIGURA NO. 132. MODELO DEL SISTEMA



Definiendo

- θ = ángulo sobre el plano XZ
- φ = ángulo sobre el plano YZ
- ψ = ángulo sobre el plano XY

Podemos simplificar el modelo del cuadcóptero en la matriz presentada en la Figura No. 133.

FIGURA NO. 133. MODELO DE LA REACCIÓN DEL SISTEMA

$$\begin{bmatrix} \ddot{Z} \\ \ddot{\theta} \\ \ddot{\varphi} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} b & b & b & b \\ -bl & 0 & bl & 0 \\ 0 & bl & 0 & -bl \\ d & -d & d & -d \end{bmatrix} * \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix}$$

Donde

- F1 es la fuerza ejercida por el motor 1.
- F2 es la fuerza ejercida por el motor 2.

- F3 es la fuerza ejercida por el motor 3.
- F4 es la fuerza ejercida por el motor 4.
- l es el largo del brazo
- b y d son constantes de las hélices y los motores.

El modelo de entrada al motor se presenta en la Figura No. 134.

FIGURA NO. 134. MODELO DE SALIDA A LOS MOTORES

$$\begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} = \begin{bmatrix} \frac{1}{4b} & 0 & -\frac{1}{2\,b\,l} & -\frac{1}{4d} \\ \frac{1}{4b} & -\frac{1}{2\,b\,l} & 0 & \frac{1}{4d} \\ \frac{1}{4b} & 0 & \frac{1}{2\,b\,l} & -\frac{1}{4d} \\ \frac{1}{4b} & \frac{1}{2\,b\,l} & 0 & \frac{1}{4d} \end{bmatrix} * \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix}$$

Donde

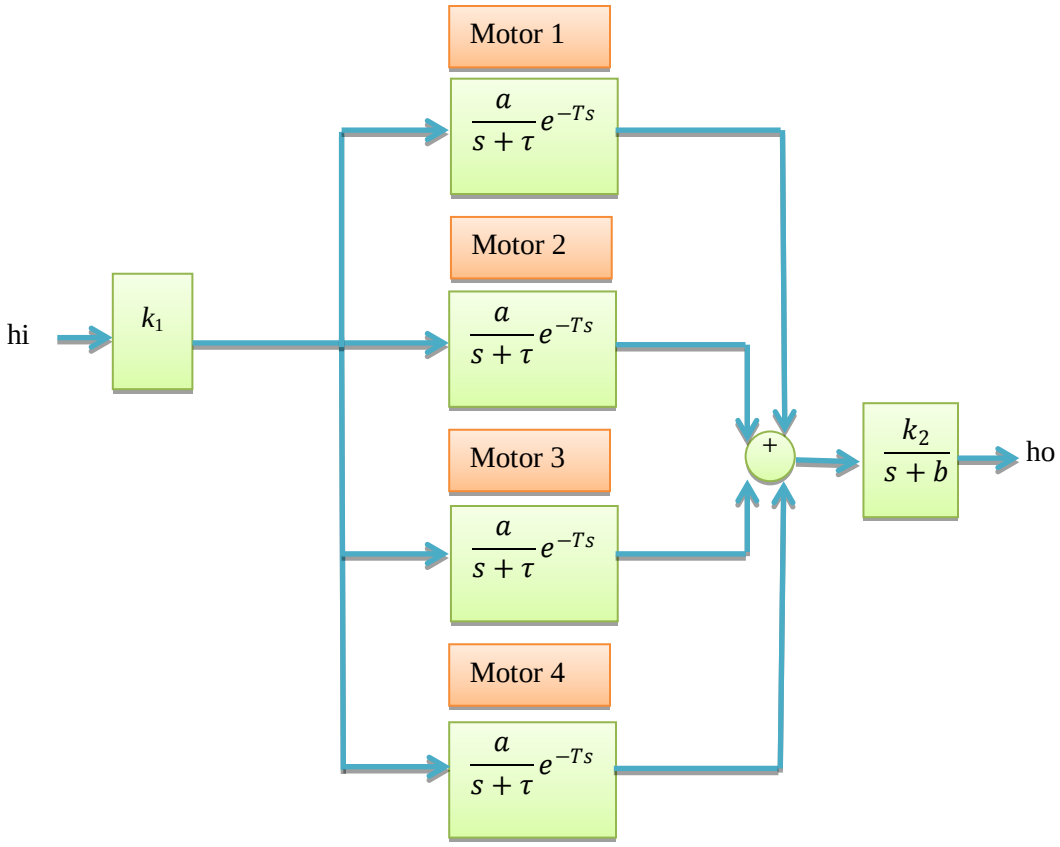
- U1 es la señal de control de Z
- U2 es la señal de control de θ
- U3 es la señal de control de φ
- U4 es la señal de control de ψ

Estos modelos se presentan simplificados, con brazos de distancias iguales, inercias despreciadas y motores similares. Se pretende que el control disminuya el efecto de estos parámetros y compense a los motores. El modelo de salida a los motores es la matriz inversa del modelo anterior. Este nos permite observar que se pueden crear varios sistemas de control independientes en parte, solo sumandos las salidas de control. Luego se creó un sistema de bloques sobre cada señal de control para poder analizarla.

Se crearon tres bloques de control. Se decidió omitir el control de altura del helicóptero porque no se contaba con la información necesaria para controlar la altura del sistema con los sensores utilizados. Esta omisión se realizó aprovechando el efecto tierra, que dice que hay mayor sustentación estando cerca del suelo. Su uso en el megaproyecto se limita a sobrevolar alturas menores a los 2 metros sobre el suelo, por lo que no es primordial un control sobre este eje de libertad. Esto puede ser mejorado. Por el momento

ese sistema se limita a un lazo abierto. En la Figura No. 135 se presenta el modelo del sistema de altura a lazo abierto, este es válido para alturas bajas.

FIGURA NO. 135. SISTEMA DE CONTROL A LAZO ABIERTO DE LA ALTURA A BAJAS ALTURAS



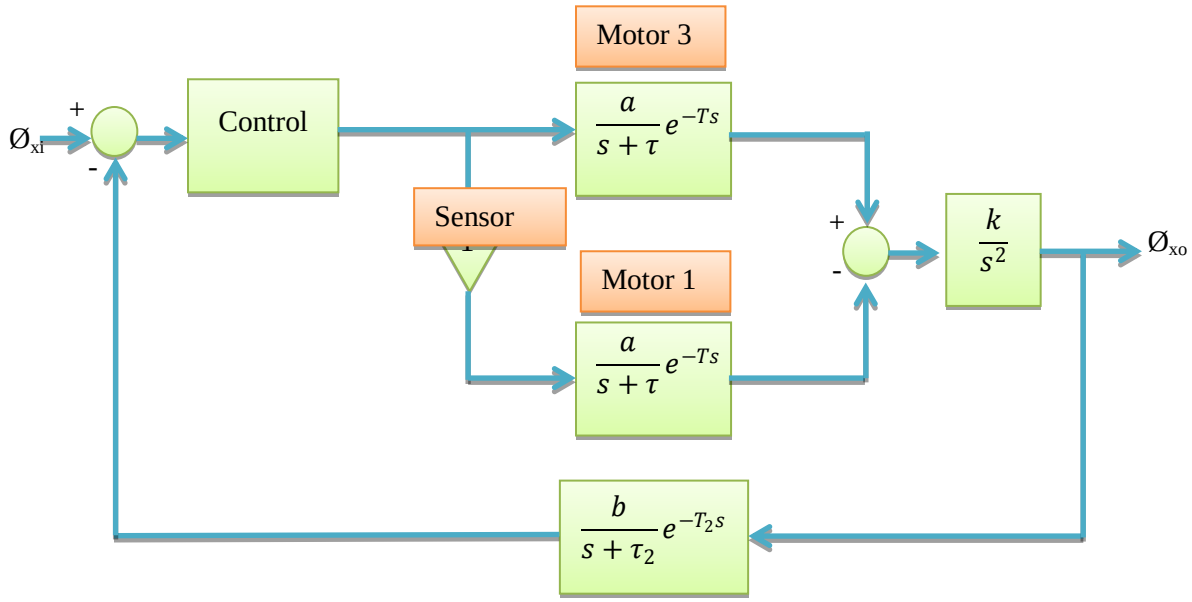
Donde

- τ es la constante de tiempo de la función de transferencia entre PWM y Fuerza
- a es su ganancia DC
- T es el tiempo de reacción de los controladores de los motores
- k 1 es la ganancia del control
- k 2 es la constante del sistema debido al efecto tierra
- hi es la altura de referencia de entrada
- ho la altura de salida

En el modelo se asume que los motores son iguales, pero al ser diferentes solamente variaría levemente los valores de las constantes. Se pretende que el control sea suficientemente robusto para responder a estos cambios.

Se realizó el modelo el sistema de control de ángulo sobre los ejes “X” y “Y” analizando el sistema. Se muestra en la Figura No. 136 el modelo obtenido sobre el eje “X”, que es similar al del eje “Y”.

FIGURA NO. 136. DIAGRAMA DE BLOQUES DEL CONTROL DEL ÁNGULO SOBRE LOS EJES "X" Y "Y"



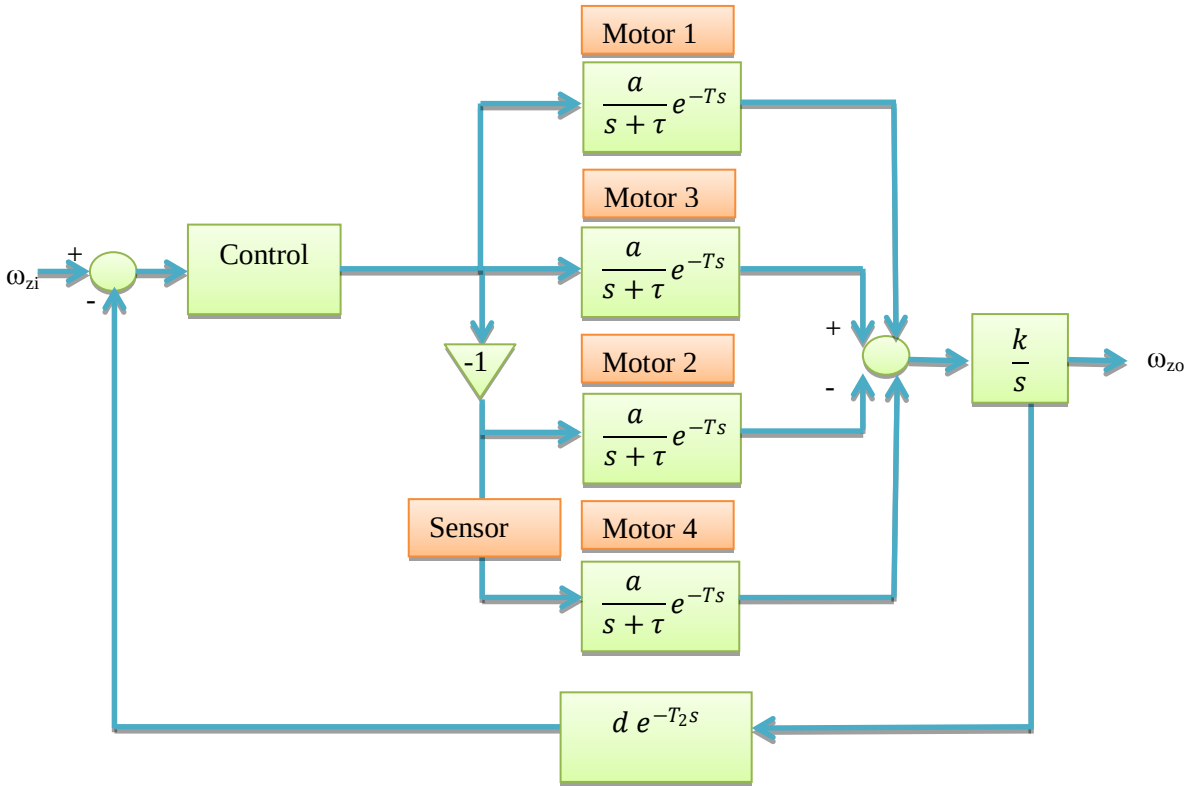
Se observa que el diagrama de bloques tiene una función $\frac{k}{s^2}$. Para evitar que se inestabilice al sistema se decide utilizar un control con una derivada de segundo orden. Esta señal de derivada se encuentra desfasada 180° de la señal proporcional, amortiguando el sistema.

Se desconocen los valores de todas las constantes del sistema, pero se conoce la forma. Esta es una de las ventajas del control PID, que no es necesario conocer todo el sistema, solo su tipo de comportamiento, y gracias a la forma se decidió utilizar un control con segunda derivada. Los parámetros del control PID fueron calibrados durante el experimento.

El control sobre el otro eje funciona de manera similar, manteniendo la congruencia de los signos y los motores.

Posteriormente se realizó el modelo el sistema de control de la velocidad angular sobre el ángulo de "YAW". El modelo se muestra en la Figura No. 137.

FIGURA NO. 137. DIAGRAMA DE BLOQUES DEL CONTROL DE VELOCIDAD ANGULAR SOBRE EL EJE "Z"

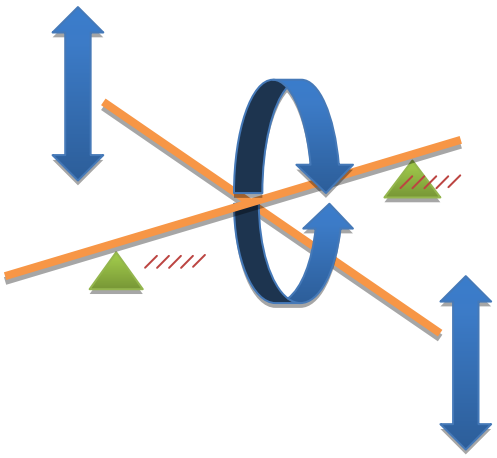


Este sistema se controló con un control PID. No se conocen todos los valores de las constantes del sistema, pero se puede calibrar más fácilmente si se terminan de calibrar los dos ejes anteriores independientemente. Esto permite una prueba más sencilla.

2. DISEÑO EXPERIMENTAL. Las pruebas de estabilización del cuadcóptero son importantes para determinar las constantes del control PID. Las pruebas se crearon para estabilizar los ejes de libertad independientemente. Se crearon dos tipos principales de pruebas para estos propósitos.

a. Estabilización angular sobre los ejes X y Y. Esta prueba se realizó dejando solo un eje de libertad en el cuadcóptero. Se calibró el ángulo sobre un eje, luego sobre el siguiente. Se varían los parámetros del PID externamente hasta obtener una respuesta satisfactoria. El esquema de esta prueba de estabilización se presenta en la Figura No. 138.

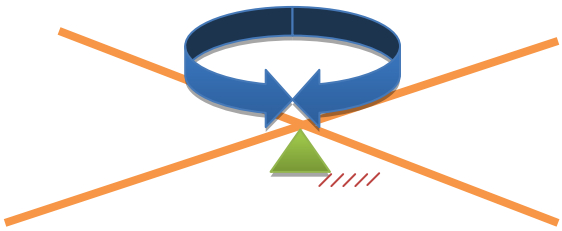
FIGURA NO. 138. ESQUEMA DE LA PRUEBA DE ESTABILIZACIÓN ANGULAR SOBRE LOS EJES



Los primeros parámetros a variar son el proporcional y la segunda derivada. Como se puede observar durante la calibración, esto nos permite mantener el sistema controlado y aumentar la velocidad de su respuesta. La parte proporcional nos mejora el tiempo de respuesta, pero puede mantener al sistema inestable. La segunda derivada nos permite amortiguarlo, disminuyendo las oscilaciones del sistema. Luego se modifican los parámetros de la integral para reducir su error en estado estable y la parte derivativa para reducirle las sobreelevaciones. Se recomienda variar los parámetros desde la comunicación serial, para poder observar su respuesta graficándola y mejorarla de una forma eficiente.

b. Estabilización de velocidad angular sobre el eje Z. Sobre el eje "Z" se suspendió el cuadcóptero en el aire, sostenido en una posición fija. Este sistema es más difícil de aislar, por lo que hay que tener mayores precauciones en sus pruebas. El esquema de esta prueba se presenta en la Figura No. 139.

FIGURA NO. 139. ESQUEMA DE LA PRUEBA DE ESTABILIZACIÓN SOBRE EL EJE Z



Este sistema presenta mayor estabilidad, por lo que solo la parte proporcional logra obtener una respuesta aceptable. Luego se modifican las ganancias derivativas e integrales para mejorarla.

Luego de calibrar los controles por separado ya se pueden realizar pruebas al aire libre.

3. RESULTADOS

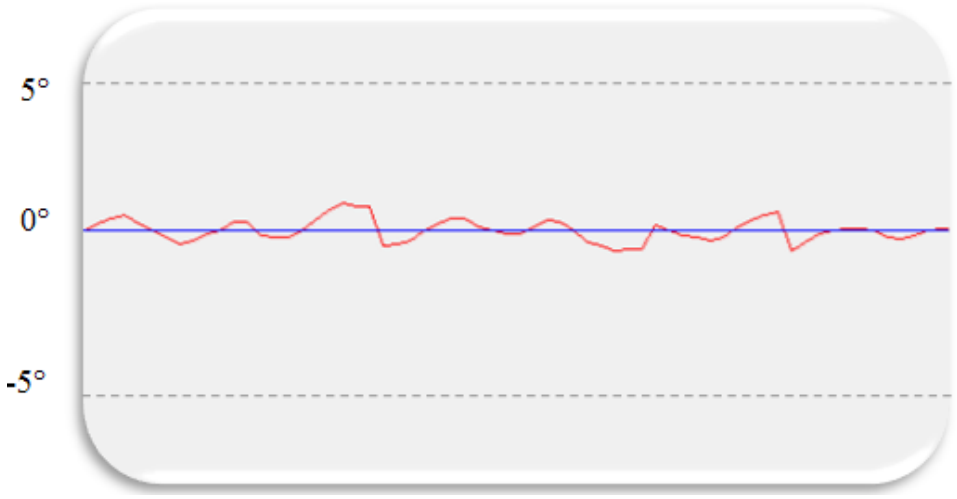
- Se cumplió con el objetivo de controlar el cuadcóptero realizando los siguientes sistemas:

FIGURA NO. 140. SISTEMAS DE CONTROL DEL CUADCÓPTERO

Sistema	Descripción
Altura	Sistema a lazo abierto haciendo uso del efecto tierra
Angular, eje X	Control PID con derivada de segundo orden
Angular, eje Y	Control PID con derivada de segundo orden
Velocidad Angular, eje Z	Control PI

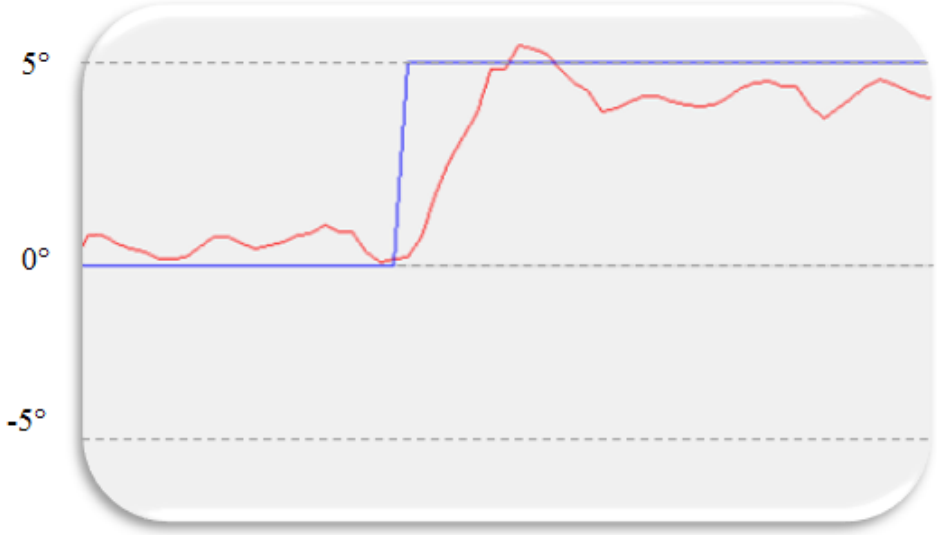
- Se obtuvo la respuesta del ángulo en estado estable mostrado en la Figura No. 141.

FIGURA NO. 141. RESPUESTA DEL ÁNGULO EN ESTADO ESTABLE



- La Figura No. 142 presenta la respuesta al escalón.

FIGURA NO. 142. RESPUESTA AL ESCALÓN



4. DISCUSIÓN. La estabilización en el vuelo de un cuadcóptero implica el uso de distintos sistemas. Para este caso en específico se implementaron tres sistemas de control. Se utilizaron controles PID por su facilidad de implementación y su flexibilidad de no necesitar de conocer todo el sistema para poder controlarlo. Este tipo de control es común en la industria y es intuitivo de calibrar.

Dentro de los controles PID se realizaron dos controles angulares sobre los ejes X y Y, y otro de velocidad angular. Estos controles se utilizar para lograr mantener al sistema en los niveles de referencia, ya que el sistema no presenta una señal estable de salida a una entrada de por ejemplo un escalón.

Para el control angular se implementó una derivada de segundo orden Para evitar que se inestabilice al sistema. El control PID de este eje tiene una derivada de segundo orden implementada, esto se debe a que el sistema nos produce una aceleración angular. Fue necesario amortiguarlo con esta segunda derivada. Se observó que el sistema se logró estabilizar utilizando solamente una parte proporcional y la segunda derivada. Este control logró amortiguar este sistema inestable por naturaleza. La implementación en cada eje no fue igual, ya que varían levemente. Debido a esto, se manejaron las constantes del PID de forma separada.

Para el control de velocidad angular se implementó un control PI. Al implementar la parte proporcional del control se obtuvo una respuesta aceptable para la aplicación, luego solo se agregó la parte integral para eliminar el error en estado estable. En la aplicación solo se utiliza este control con tres niveles de referencia: giro a la derecha, giro a la izquierda y sin giro (velocidad angular = 0).

El sistema de altura se implementó a lazo abierto, utilizando el efecto suelo. Este efecto nos dice que a una altura baja se obtiene una sustentación mayor, proporcional a la distancia del suelo. Una implementación posible es utilizando un sensor ultrasónico hacia el suelo.

A la aplicación de todos los sistemas no logra obtener una respuesta ideal, sin oscilación, debido a varios problemas específicos. Entre estos la oscilación de ángulos pequeños presente en el sensor de ángulo, los largos tiempos de respuesta en los motores y sus drivers.

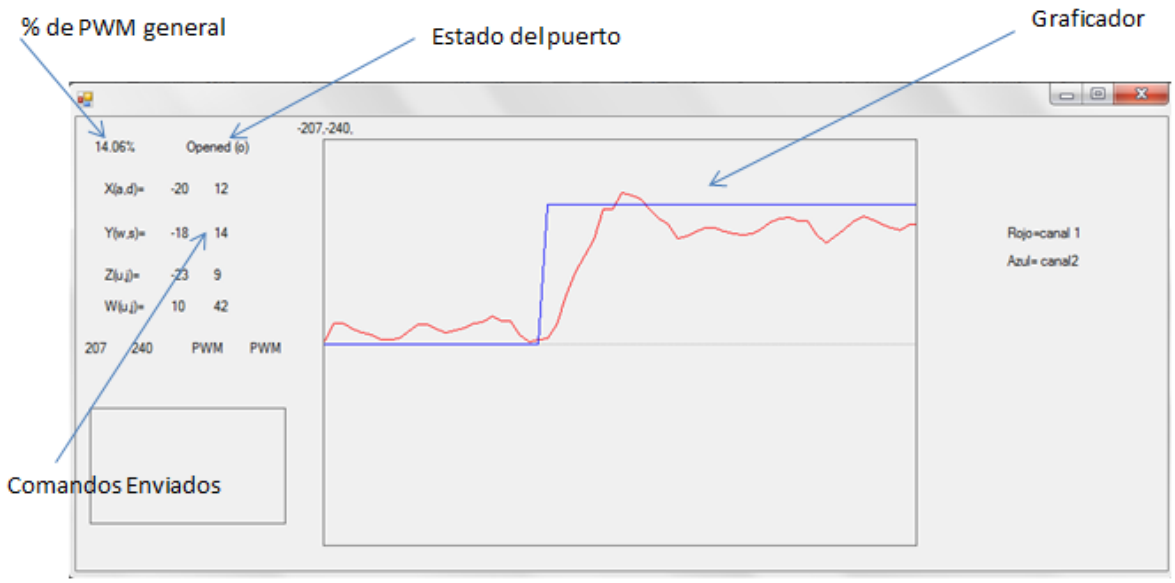
E. CONTROL MANUAL DESDE LA COMPUTADORA

1. DISEÑO DEL PROGRAMA. Se elaboró un programa en la computadora con la herramienta Microsoft Visual C# para depurar errores y para crear un ejemplo de control manual del cuadcóptero desde la computadora. Éste debía tener un campo en el que mostrara datos enviados desde el cuadcóptero, como inclinaciones y de enviar comandos al mismo. Para un mejor control del helicóptero se programó una interfaz con un control de computadora, utilizando Microsoft Visual C# por su facilidad y su entorno de programación.

2. RESULTADOS

- La interfaz para el programa de control manual se muestra en la Figura No. 143.

FIGURA NO. 143. INTERFAZ DE CONTROL Y DEPURACIÓN



- La comunicación inalámbrica se lleva a cabo utilizando un Xbee con un adaptador a USB, que nos permitía la fácil comunicación con este utilizando un puerto COM virtual.

- El mando de control se muestra en la Figura No. 144:

FIGURA NO. 144. MANDO DE CONTROL

3. DISCUSIÓN. El programa facilitó la depuración de los errores en el código. Se graficaron dos datos recibidos en formato ASCII divididos por comas. Esto también permite que se puedan leer sus datos examinarlos en Excel.

El control fue de gran ayuda para controlar manualmente el cuadcóptero, pues su movimiento resulta intuitivo. Este modifica los niveles de referencia del vehículo. Se puede modificar para agregarle nuevas funciones. La gráfica a tiempo real también es de gran ayuda en la calibración del sensor y del control PID.

El programa puede ser modificado para ser mejorado, pero hay que recordar que su propósito principal es la depuración de errores y un control manual básico. Se utilizaron las librerías de DirectX para obtener la lectura del control, este facilito su interfaz. El envío de datos a la computadora se realizó por medio de codificación ASCII, y la comunicación hacia el microcontrolador se realizó por comandos. Estos comandos definidos del helicóptero se utilizaron para controlarlo inalámbricamente.

F. DISEÑO DE CIRCUITO DETECTOR DE OBJETOS

1. DISEÑO EXPERIMENTAL. Se diseñó un circuito basado en comparadores con amplificadores operacionales y sensores ultrasónicos. Para ello se establece la de distancia mínima a la que el helicóptero puede estar de un objeto antes que el circuito dispare la señal de emergencia para que algún sistema de control identifique que existe peligro de chocar. El sensor ultrasónico proporciona una salida analógica que representa la distancia, el amplificador operacional compara el voltaje de la salida del sensor con una señal de voltaje fija y determina si hay peligro o no.

2. CÁLCULOS Y RESULTADOS. La relación entre capacitor y resistencia para el filtro se determina por medio de la frecuencia de corte requerida, que para nuestro sistema se establece como 3Hz. La ecuación que establece la frecuencia de corte para un sistema RC está dada por la siguiente expresión.

$$\omega c = 2\pi f = \frac{1}{RC}$$

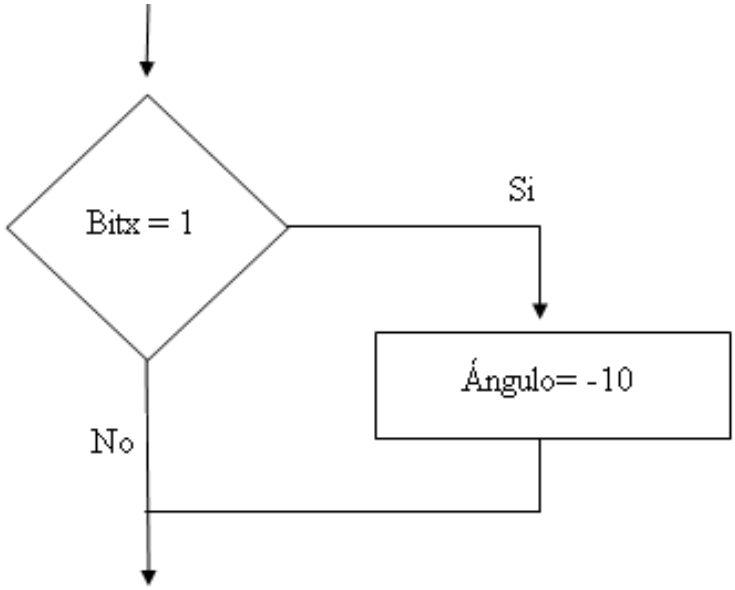
Para los valores disponibles en el mercado se establecen los siguientes valores de capacitor y resistencia.

$$R = 25k\Omega, C = 2.2\mu F$$

Que resulta en una frecuencia de 2.89Hz, lo cual se considera aceptable.

Para la implementación del circuito detector de objetos, se requiere haber establecido un control de inclinación. Cuando el circuito detecte objeto colocará su salida respectiva en un “1” lógico y cuando no un “0” lógico. De esta manera, cuando exista objeto cercano el controlador es capaz de identificarlo y establece un ángulo fijo que gire el vector de sustentación contraria al movimiento, y se cree una desaceleración del cuadcóptero. El algoritmo de programación se muestra en la Figura No. 145.

FIGURA NO. 145. ALGORITMO PARA LA IMPLEMENTACIÓN DEL CIRCUITO DETECTOR DE OBJETOS



3. DISCUSIÓN. Una frecuencia de corte de 3Hz es relativamente baja, la razón es porque resulta importante que nuestra medición de distancia sea lo suficientemente estable para que el cuadcóptero identifique con certeza si existe peligro o no. Dado que el sistema es dinámico, es decir, que se está moviendo mientras se toman mediciones y además el sensor ultrasónico es afectado por la velocidad del viento, el ángulo con que choca la onda en una superficie y las características físicas de dicha superficie, la lectura de distancia está sujeta a mucho ruido. Esta variación indeseable en la lectura, se disminuye diseñando el filtro pasa baja implementado con una frecuencia de corte de 3Hz. Es importante notar que entre más se filtre una señal se producirá mayor retraso en la medición, por lo tanto, se debe establecer un filtro lo suficientemente robusto como para quitar ruido sin que produzca mayor retraso en la señal.

IX. SENSOR E INTERFAZ RS-232

Este capítulo contiene una síntesis del módulo de sensor e interfaz RS-232. Para información más detallada del módulo, consultar el trabajo *Implementación de una interfaz de datos inalámbrica RS-232 para el sensor laser de distancia Hokuyo URG-04LX-UG01* [13].

El módulo consiste en la implementación de la primera etapa del sistema de localización del robot. El módulo debe encargarse de obtener los datos de distancia necesarios desde el sensor laser para que el robot sea capaz de tomar decisiones, evadir obstáculos y explorar ambientes desconocidos.

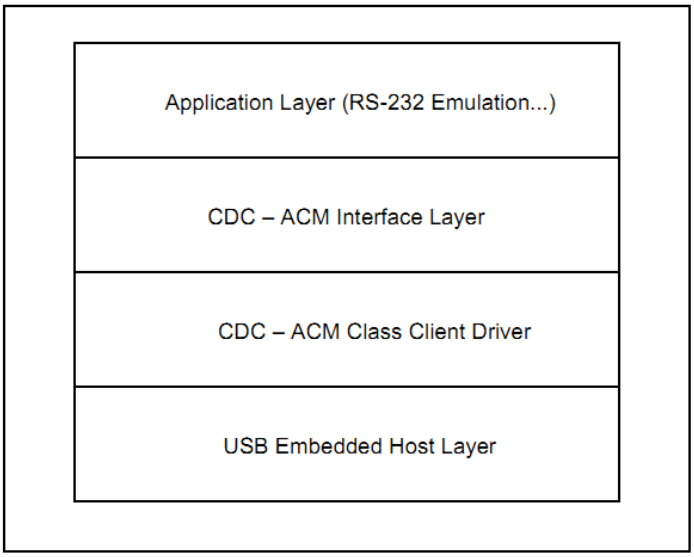
Debido a la naturaleza del sensor laser (dispositivo USB) fue necesario implementar un controlador USB en un microcontrolador para poder obtener los datos de distancia inalámbricamente.

Para implementar el controlador del sensor laser se utilizó un microcontrolador de 16 bits de la familia PIC24 de Microchip (PIC24FJ64GB002) y su módulo USB 2.0 OTG (On-The-Go) que es compatible con la clase USB del sensor láser (Communication Device Class). Se utilizó la librería USB Firmware desarrollada por Microchip; esta forma parte de la librería de aplicaciones para la familia de microcontroladores PIC24F y dsPIC33.

Posteriormente, se decodificaron y procesaron los datos en una aplicación desarrollada en un lenguaje de alto nivel (C#) que implementa el protocolo de comunicación desarrollado por el fabricante del sensor SCIP en su versión 2.0.

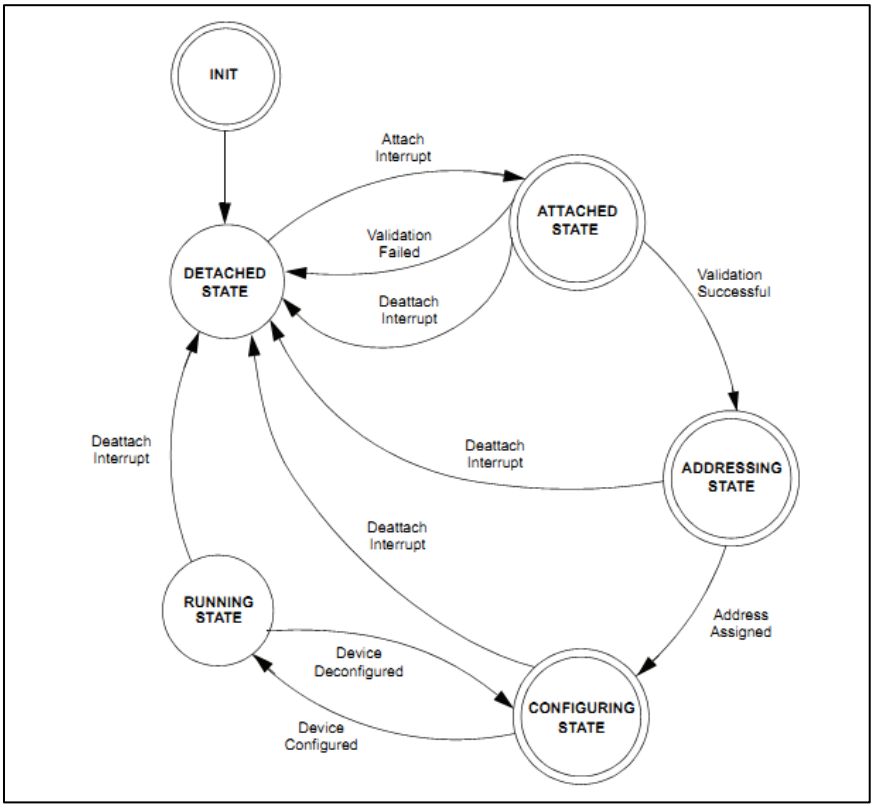
La Figura No. 146 muestra la estructura del controlador USB implementando en el microcontrolador utilizando la librería USB Firmware.

FIGURA NO. 146. ESTRUCTURA DEL CONTROLADOR USB IMPLEMENTADO. [19]



La Figura No. 147 muestra el diagrama de estados del controlador USB implementado en el microcontrolador mencionado.

FIGURA NO. 147. DIAGRAMA DE ESTADOS DEL CONTROLADOR USB IMPLEMENTADO. [19]



X. MAPEO Y EXPLORACIÓN

En este capítulo se desarrolla el trabajo elaborado en el trabajo *Vuelo automatizado* [47]. El orden de las secciones es el orden cronológico de las etapas en que se elaboró el mismo.

A. SIMULACIÓN

En el desarrollo de cualquier proyecto, los costos tienen un factor decisivo en la metodología de trabajo. Siempre se busca reducir a un mínimo los costos, y por ende se busca reducir los riesgos. Debido a que el megaproyecto SEEQ se basa en un vehículo aéreo, realizar pruebas y trabajar sobre este vehículo en las etapas iniciales hubiera sido una decisión inapropiada, pues se hubiera puesto en juego la salvedad del sensor, además de otros elementos involucrados.

Esta situación, y el desarrollo en paralelo de todos los módulos del megaproyecto, hizo necesario crear un ambiente de experimentación y desarrollo, en que se pudieran realizar pruebas sin riesgos y sin costos adicionales. La solución fue crear una simulación del megaproyecto.

1. DISEÑO EXPERIMENTAL. Esta simulación se desarrolló en el lenguaje de programación C#, debido a las múltiples ventajas y simplicidad que ofrece.

La idea de la simulación fue poder representar al robot, al ambiente y al sensor de una manera simplificada pero fiel al comportamiento real de los mismos, para de esta manera lograr aplicar sobre sus resultados los algoritmos involucrados en el módulo de vuelo automatizado.

La representación del robot, simplemente se modeló como un círculo en el espacio, el cual se mueve de acuerdo a las coordenadas cartesianas que se le indiquen.

El ambiente, fue representado por medio de dos imágenes generadas en la herramienta de edición de imágenes PAINT. En ambos ambientes se trató de incorporar formas y polígonos complicados, de manera que los algoritmos desarrollados posteriormente fueran robustos y pudieran responder ante casi cualquier situación. El espacio libre en los ambientes, se estableció con color blanco, y los obstáculos y objetos en el ambiente se representaron con color negro.

El sensor, tuvo la representación más compleja en la simulación. Se buscó que el modelo utilizado cumpliera con las mismas características que el sensor real, por lo que la función de escanear en la aplicación, obtiene datos del ambiente simulado en el mismo formato, y de la misma manera que el sensor real.

El sensor del megaproyecto, Hokuyo URG-04LX UG01, entrega una cantidad de datos variables, pero tiene un rango angular limitado a 240 º. Esto quiere decir que se puede ajustar su resolución, variando así la cantidad de puntos en los que divide los 240 º. En la etapa inicial de desarrollo del proyecto, la resolución establecida fue de 682 datos, teniendo una separación entre datos de aproximadamente 0.35 º. La distancia que representa cada punto, tiene un máximo de 4095 milímetros para ser considerado como un punto válido. Esta distancia máxima es ajustable, y para el proyecto se dejó en 4095 milímetros. La distancia mínima es 20 milímetros. Con este rango de distancias válidas, se hace una regla de verificación, y cualquier dato que no sea considerado como válido, se guarda en el listado como un -1. Todos los datos con valor -1 en el vector de datos del sensor, son ignorados en el procesamiento y análisis posteriores.

En la simulación, para simplificar el manejo de datos, se escaló el rango de validez a un máximo de 150.

2. RESULTADOS

FIGURA NO. 148. AMBIENTE DE SIMULACIÓN 1

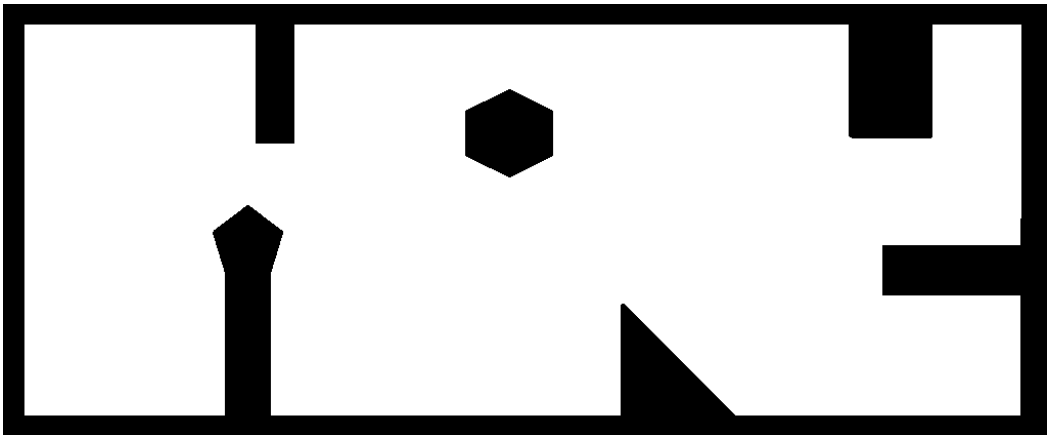


FIGURA NO. 149. AMBIENTE DE SIMULACIÓN 2

FIGURA NO. 150. MODELO DEL ROBOT EN AMBIENTE 1

FIGURA NO. 151. REPRESENTACIÓN GRÁFICA DE RESPUESTA DEL SENSOR SIMULADO

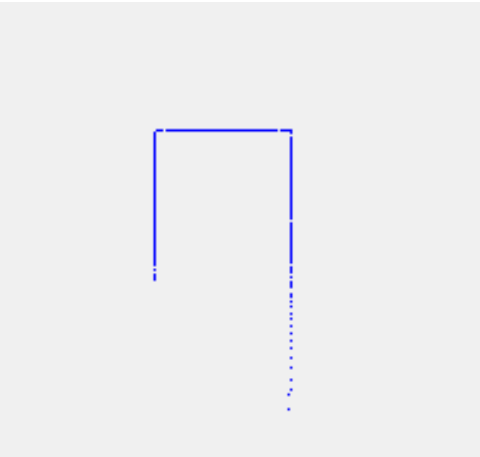


FIGURA NO. 152. INTERFAZ INICIAL DEL PROGRAMA DE SIMULACIÓN

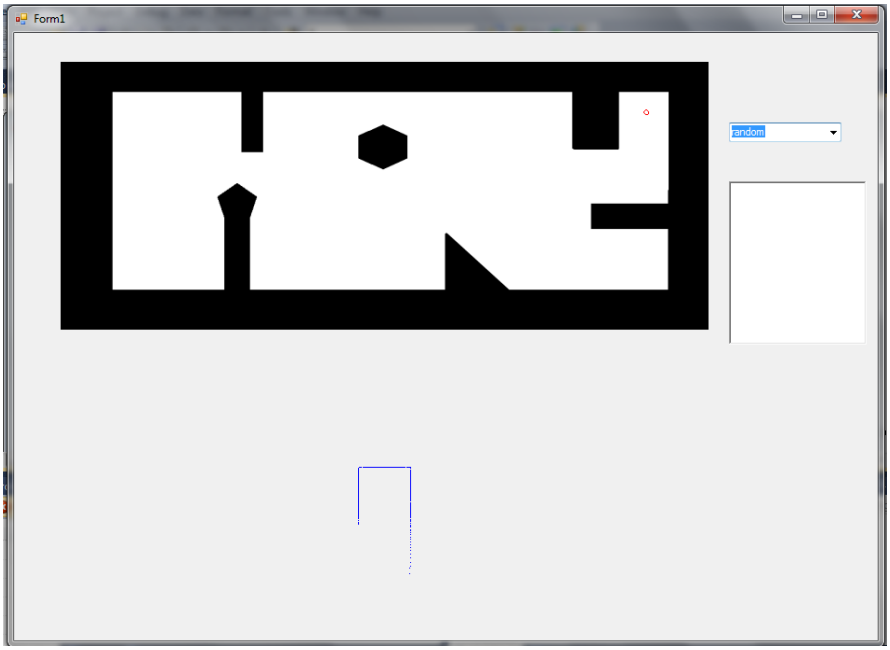
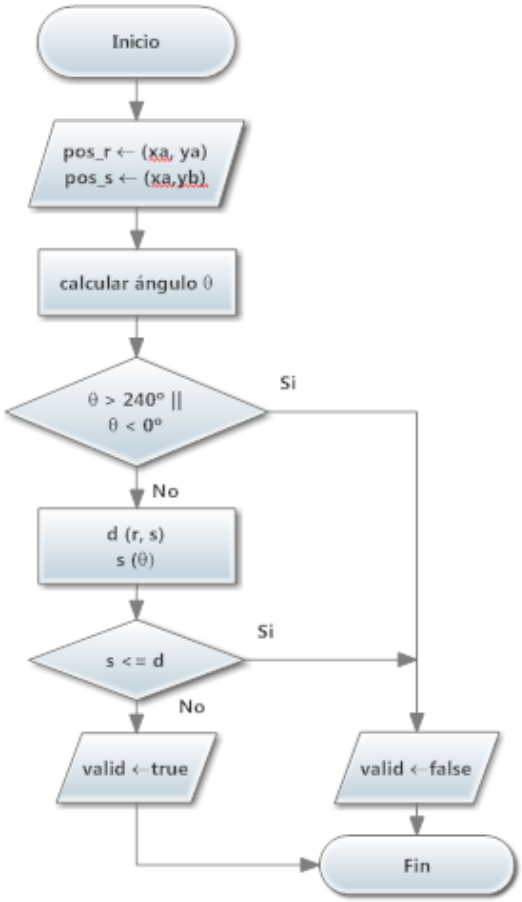


FIGURA NO. 153. DIAGRAMA DE FLUJO DEL ALGORITMO DE VERIFICACIÓN DE TRAYECTORIA



3. **DISCUSIÓN.** Como se puede ver en las Figuras No. 148 y 149, los ambientes de simulación contienen distintas figuras geométricas y polígonos, que pretenden asemejar un ambiente real, y las dificultades que se podrían presentar en este. El objetivo del programa de simulación fue poner a prueba los algoritmos desarrollados posteriormente. Una vez se logró trabajar todos los algoritmos de manera exitosa en los ambientes simulados, se procedió a la implementación real.

El sensor simulado, toma en cuenta la limitante de 240º de libertad, y se estableció una resolución de 683 datos por escaneado. Estos datos se obtienen en coordenadas polares, simplemente como un arreglo de distancias en el rango de 0 a 150, y -1 en los puntos inválidos. El ángulo correspondiente a cada punto se calcula posteriormente, junto con las coordenadas (x, y) del mismo. Estas coordenadas se manejan con respecto a la posición del robot.

Para hacer los más real posible la simulación, se agregó el algoritmo de detección de paredes y obstáculos. Este algoritmo evita que el robot atraviese paredes, y que se mueva a espacios dentro de las paredes u obstáculos.

La primera etapa del algoritmo revisa que la posición a la que se pretende mover al robot, no sea un punto negro. Si esto se cumple se realiza el movimiento deseado, si no, vuelve a generar una posición para el movimiento hasta llegar a una válida. La segunda etapa, revisa que la trayectoria del robot no atraviese paredes u objetos. La lógica de esta etapa, se plasma en el diagrama de flujo de la Figura No. 153.

El diagrama muestra que el primer proceso es encontrar el ángulo entre la posición actual y la siguiente posición del robot. Este ángulo se debe calcular debido a que en la simulación, los movimientos del robot se dan en coordenadas cartesianas. Si el ángulo esta fuera del alcance del sensor, se toma como invalida la siguiente posición y termina el algoritmo. Si está en el intervalo válido, se utiliza el ángulo para encontrar su distancia correspondiente en el vector de distancias obtenido con el sensor, y se compara la distancia del sensor, y la distancia entre posición actual y siguiente posición. Si la distancia del sensor es menor a la distancia de la siguiente posición, significa que existe un obstáculo en la trayectoria, y el movimiento es inválido. Si no se cumple ninguna de las condiciones, la trayectoria es válida.

Con este algoritmo, y los modelos planteados para la simulación, se formó una buena plataforma para probar los algoritmos de generación de mapas y exploración. Solo un aspecto quedaba por contemplar. En un ambiente simulado, el comportamiento de los elementos a simular se puede controlar de manera completa. Se le puede dar las propiedades deseadas a cualquier modelo, sin embargo en la vida real, el comportamiento de un sistema es inexacto, y el ruido juega parte importante en su respuesta.

Para modelar el ruido en la simulación, se determinó que la fuente de error de mayor influencia corresponde al movimiento del robot. Un robot real, se basa en técnicas de odometría para determinar la cantidad de movimiento que ha realizado, sin embargo el dato de la odometría no es 100% exacto, por lo que se introducen errores en cada movimiento. Para simular este error de movimiento, se consideró la variable de estado

que describe la posición del robot en cualquier momento. Esta posición está dada con respecto al último mapa local generado, y se describe por medio de un valor de traslación en el eje horizontal (x), un valor de traslación en el eje vertical (y), y un valor de rotación (θ) sobre sí mismo. Para introducir ruido a esta variable de estado, se genera un valor aleatorio limitado para cada componente, es decir tres ruidos aleatorios que alteran el estado teórico del robot. Los errores de traslación, se limitan a ± 5 unidades en la escala trabajada, correspondiente a un valor de 136 mm en la realidad. El error de rotación, se limita a un máximo de 10 º.

Este ruido permite desarrollar los algoritmos de una manera más robusta y más funcional en un ambiente real.

B. ALGORITMOS PARA GENERACIÓN DE MAPA

En esta sección, se desarrolla la lógica base del algoritmo de construcción del mapa. Este algoritmo, es una etapa esencial del trabajo *Vuelo automatizado*, y es la base para la exploración autónoma, pues se necesita un buen conocimiento del ambiente para que el robot pueda tomar las decisiones correctas.

La función de los algoritmos de generación del mapa es construir una representación del ambiente en el que se encuentra el robot o explorador, en base a la información obtenida con sensores. En el caso específico del megaproyecto SEEQ, el sensor con el que se cuenta es un sensor laser, de manera que la información recolectada consiste en una lista o arreglo de puntos, que corresponden a la distancia radial desde el robot hasta un obstáculo u objeto en el ambiente, a distintos ángulos distribuidos de una manera uniforme.

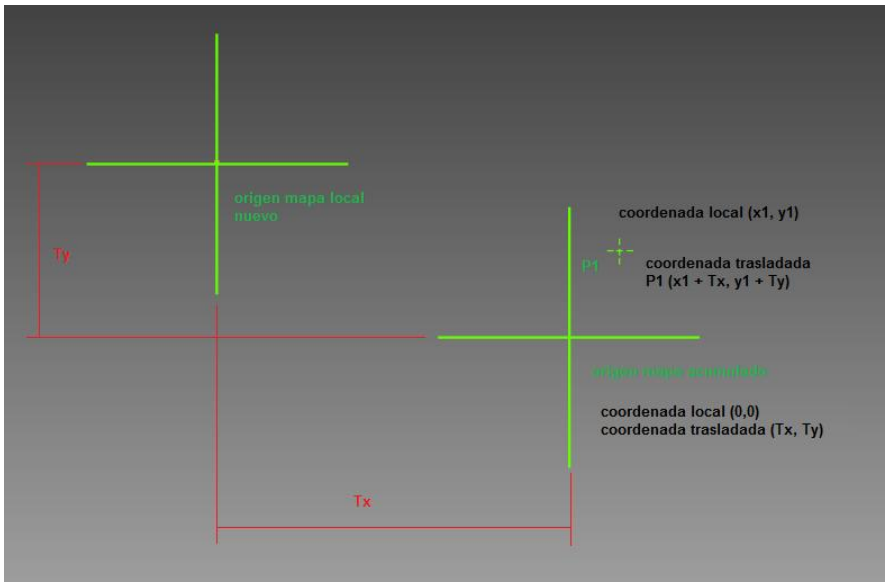
Una vez obtenido un set de datos, estos son considerados como un mapa local, teniendo como origen la posición del robot. Este mapa local se obtiene en coordenadas polares (r, θ) como ya se mencionó, pero se genera un mapa en coordenadas cartesianas (x, y) equivalente para simplificar el procesamiento de datos en etapas posteriores.

Conforme se va moviendo el robot, se van obteniendo nuevos mapas locales. Esto implica que con cada movimiento, se debe ajustar el mapa anterior al nuevo origen, formándose así un mapa local y un mapa acumulado para cada movimiento. Para ajustar el mapa acumulado al local nuevo, el primer paso es trasladar todas las coordenadas al nuevo origen. Luego se entra a la etapa de alineamiento y coincidencia, y finalmente se realiza la unión de ambos mapas para formar un acumulado global.

1. ALGORITMO DE CAMBIO DE COORDENADAS

a. Diseño Experimental. El algoritmo de cambio, o algoritmo de traslación de mapa acumulado, se basa en matemática simple, y se ejecuta punto a punto. La base matemática del algoritmo se plasma en la siguiente figura:

FIGURA NO. 154. CAMBIO DE COORDENADAS



La idea del algoritmo es que el origen del mapa acumulado, se cambia a una coordenada con respecto al nuevo origen, teniendo como valores el cambio o traslación en cada eje. Este cambio de valor del origen del mapa acumulado, ocasiona que las coordenadas de todos los puntos se ajusten al nuevo origen. La aplicación de la traslación, resulta más manejable utilizando los mapas en coordenadas cartesianas.

Una vez trasladado el mapa en coordenadas cartesianas, se recalculan las coordenadas polares de cada punto, y se actualiza sus valores. En este momento se cuenta con un mapa acumulado actualizado con el último movimiento del robot.

b. Resultados

FIGURA NO. 155. POSICIÓN DE ESCANEO EN AMBIENTE DE SIMULACIÓN 1

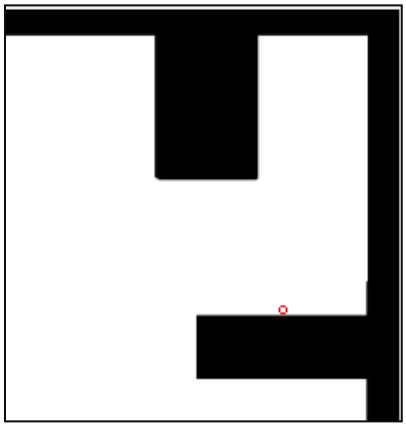


FIGURA NO. 156. MAPA LOCAL OBTENIDO POR SENSOR

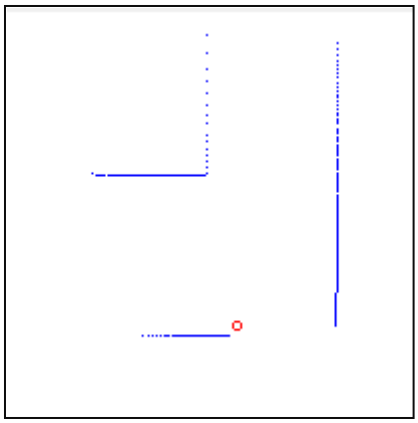


FIGURA NO. 157. APLICACIÓN DE ALGORITMO DE CAMBIO DE COORDENADAS (CASO IDEAL)

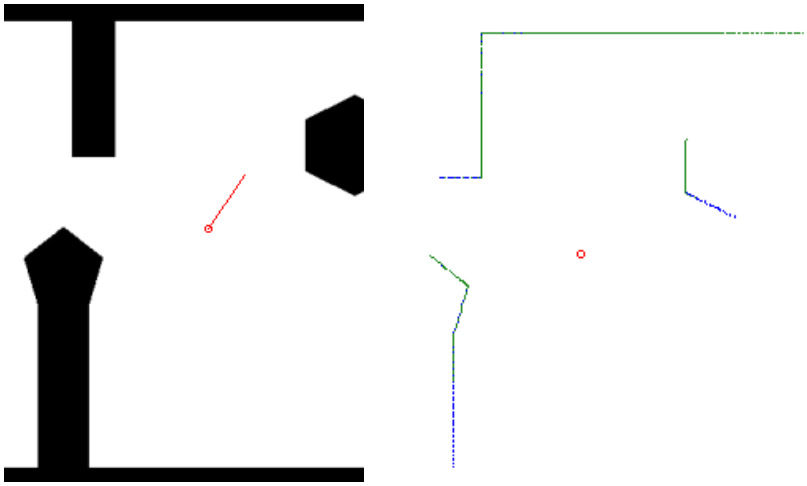
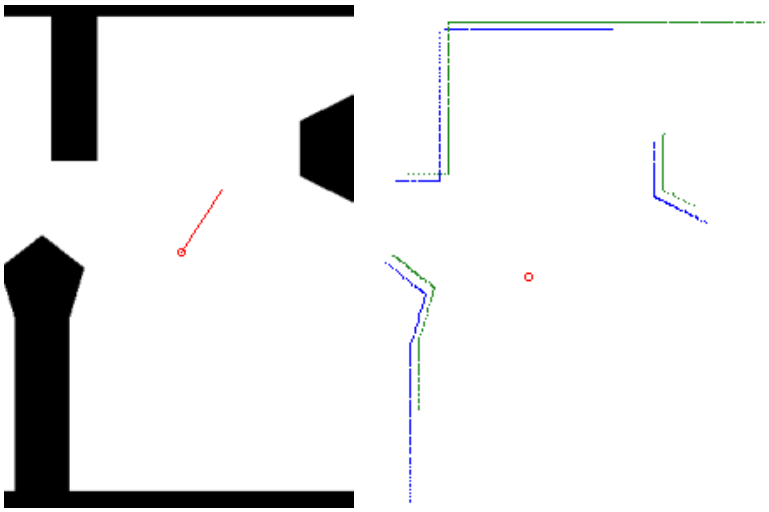


FIGURA NO. 158. APLICACIÓN DE ALGORITMO DE CAMBIO DE COORDENADAS (CASO REALISTA)



c. Discusión. En la Figura No. 156, se puede observar una representación gráfica de un mapa local generado en la simulación. Cada mapa local corresponde directamente con los datos obtenidos del sensor, tomando en consideración únicamente los datos válidos.

El valor de la traslación con respecto al nuevo origen, corresponde al movimiento del robot hasta su siguiente posición. La obtención de este valor se va a discutir en secciones posteriores, por el momento se va a asumir que ya se cuenta con esta información.

Se consideraron dos casos de análisis de movimiento para mostrar la necesidad de los siguientes algoritmos a desarrollar. El primer caso, es el caso ideal, en el cual el movimiento ejecutado por el robot, corresponde exactamente con el dato proporcionado al algoritmo de cambio de coordenadas. Este caso se muestra en la Figura No. 157. En la imagen de la izquierda se muestra el movimiento ejecutado en el ambiente de simulación 1, y en la derecha se muestra el mapa local nuevo (color azul), y el mapa acumulado trasladado (color verde). Se puede observar que ambos mapas, con el mismo punto de referencia para sus coordenadas, coinciden de manera perfecta. Los puntos observados desde ambos mapas se traslapan correctamente, y los que no, siguen el contorno del ambiente de manera correcta.

En este caso, la generación del siguiente mapa acumulado podría realizarse simplemente almacenando todos los puntos de ambos mapas en un mismo arreglo. Sin embargo, este caso es muy difícil de alcanzar, si no es que imposible, debido a que los sensores disponibles en la actualidad no tienen las características necesarias para dar una respuesta tan exacta. Además, los sensores que proveen una mejor respuesta, son también los más caros. Para el problema planteado en el megaproyecto SEEQ, resultaría incorrecto considerar este caso como base de desarrollo, pues si con vehículos terrestres es difícil alcanzar exactitudes altas en datos de movimiento, con vehículos aéreos resulta aún más complicado.

El segundo caso, el realista, es el más apropiado para el megaproyecto SEEQ. En este caso se toma en cuenta el ruido implementado en la simulación para darle más realismo a sus resultados. Se puede observar en la Figura No. 158 como al ejecutarse un movimiento, el mapa acumulado trasladado con el algoritmo de cambio de coordenadas (color verde) está desalineado con el mapa local nuevo (color azul). A pesar de que en teoría tienen el mismo punto de referencia (origen), el ruido proveniente de la medición del movimiento desajusta el mapa anterior, y se hace evidente que no es suficiente el algoritmo de cambio de coordenadas para construir una buena representación del ambiente. Si solo se utilizara éste algoritmo, el error se iría incrementando con cada movimiento, de manera que el mapa final no sería una representación válida del ambiente.

2. ALGORITMO DE ALINEAMIENTO Y COINCIDENCIA.

Como se planteó en la sección anterior, considerando el caso realista para el movimiento del robot, no es suficiente el algoritmo de cambio de coordenadas para unir de manera correcta un mapa local con el mapa acumulado anterior. Alguna técnica de corrección se hace necesaria para corregir los errores introducidos al análisis por los sensores e inexactitud del movimiento del robot.

Estas técnicas de corrección, son usualmente conocidas como algoritmos de alineamiento y coincidencia, pues su meta es lograr ajustar dos conjuntos de información de la mejor manera posible, a través de alinear y hacer coincidir datos entre los conjuntos.

Durante la etapa de investigación del megaproyecto, se determinó que estos algoritmos son parte de los estudios de especialización en el campo de la inteligencia artificial, y existen muchas soluciones propuestas al problema planteado, sin embargo mucha de la teoría que respalda a estas soluciones está fuera del alcance de una licenciatura, tales como los trabajos [42] y [57]. Esto llevó a realizar una investigación más profunda en búsqueda de una solución manejable pero robusta, y finalmente llevó a la teoría presentada a continuación.

a. Diseño experimental

1) Antecedente de algoritmo. En la investigación *Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans* [29], los autores plantean dos soluciones al problema de alineamiento y coincidencia, considerando estas técnicas como una solución al problema de localización. Si se logra alinear correctamente el mapa local nuevo del robot, con el mapa acumulado anterior, se obtendría un dato de la traslación y rotación necesaria para este ajuste, y estos datos, junto con el conocimiento del desplazamiento teórico del robot hasta la nueva posición, permiten al robot localizarse de manera exacta en el mapa. Los algoritmos presentados representan una técnica de reducción de error en la unión de mapas locales para formar un mapa completo.

Los autores, plantean el problema de localización para un robot terrestre de la siguiente manera:

> «Suponga que el robot empieza en la posición P_{ref} (posición de referencia) y toma un escaneo S_{ref}. El robot se mueve en un ambiente estático a una nueva posición P_{new} y toma otro escaneo S_{new}. La diferencia aproximada de P_{new} desde P_{ref} (es decir la traslación relativa y rotación) es conoce generalmente de la información de odometría. Sin embargo esta información es generalmente imperfecta debido al deslizamiento de las llantas. La labor es determinar la diferencia exacta entre P_{new} y P_{ref} alineando los dos escaneos.
>
> El problema de coincidencia se formula de la siguiente manera: Inicialmente se asume que la posición de S_{new} es P'_{new}. Se debe encontrar una rotación ω y una traslación T para S_{new} de manera que, tras aplicar la transformación, S_{new} esté alineado con S_{ref}. » [29]

De acuerdo con los autores, existen dos tipos de discrepancias que no permiten alinear perfectamente dos sets de datos, adicionales a la ya considerada en este trabajo. La primera corresponde a las pequeñas desviaciones de los datos obtenidos por el sensor, con respecto a las distancias reales con el ambiente correspondiente a ruido aleatorio. Estos errores se pueden caracterizar por una función de distribución de error, que depende del sensor utilizado. La segunda discrepancia corresponde a la oclusión, que sucede cuando en un set de datos se cuenta con información de un segmento del ambiente, y en el otro no. Estos segmentos de discrepancia por oclusión no se deben considerar dentro de los algoritmos. [29]

El enfoque que se le da a los algoritmos de alineamiento y coincidencia propuestos en el estudio, se reduce a plantear una función de distancia entre la escaneada de referencia (que en este trabajo es el mapa acumulado previamente trasladado a la nueva referencia) y la escaneada nueva (que corresponde al mapa local nuevo). Luego se busca reducir a un mínimo esta función de distancia ajustando la escaneada de referencia. La función de distancia considera los tres parámetros que describen la posición del robot (x, y, θ).

El primer algoritmo presentado fue denominado por sus autores como “Algoritmo de Coincidencia por búsqueda /mínimos cuadrados”. La idea es computar las direcciones tangentes en cada escaneado a través del ajuste de líneas a sets de puntos consecutivos.

Luego se asocian correspondencias aproximadas de puntos y se formula una ecuación lineal de la traslación desconocida T. Luego usando los pares de correspondencia, se define un modelo de mínimos cuadrados para T el cual también representa una distancia de coincidencia como función de ω. Los puntos con oclusión, se detectan usando compuertas o filtros. Luego el paso final consiste en buscar una rotación ω que minimice la función de distancia. La traslación se resuelve usando el método de mínimos cuadrados. [29]

Lo más importante de este primer algoritmo, y que se utilizó como base para desarrollar parte del algoritmo de alineamiento trabajado, es el algoritmo de proyección. La proyección es un aspecto importante para obtener buenos resultados en el alineamiento. Su función es omitir todos los puntos que no se consideran relevantes para el algoritmo, pues estos puntos introducen ruido y distorsionan los resultados.

El segundo algoritmo, es la contribución principal del estudio para este trabajo. El "Algoritmo de Coincidencia por Correspondencia de Puntos", es un algoritmo iterativo que se basa en la correspondencia punto a punto. La idea del algoritmo es que para cada punto P_i en S_{new}, se usa una regla sencilla (que es independiente de la rotación y traslación actual) para determinar un punto correspondiente P'_i en S_{ref}. Luego de todas las parejas de puntos correspondientes, se computa una solución de mínimos cuadrados de la rotación y traslación relativas. Esto reduce el error de posición entre los dos mapas, y se repite el proceso hasta que el algoritmo converja. [29]

La solución de mínimos cuadrados, se deriva de minimizar la función de distancia propuesta a continuación:

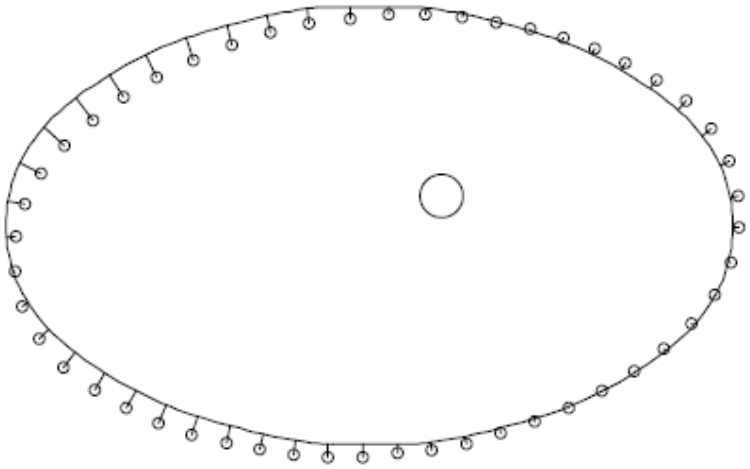
FIGURA NO. 159. FUNCIÓN DE DISTANCIA A MINIMIZAR

$$E_{dist}(\omega, T) = \sum_{i=1}^{n} |R_\omega P_i + T - P'_i|^2$$

Para aplicar en la práctica esta función, se presenta en el estudio una forma cerrada de la misma (apéndices de mapeo y exploración). [29]

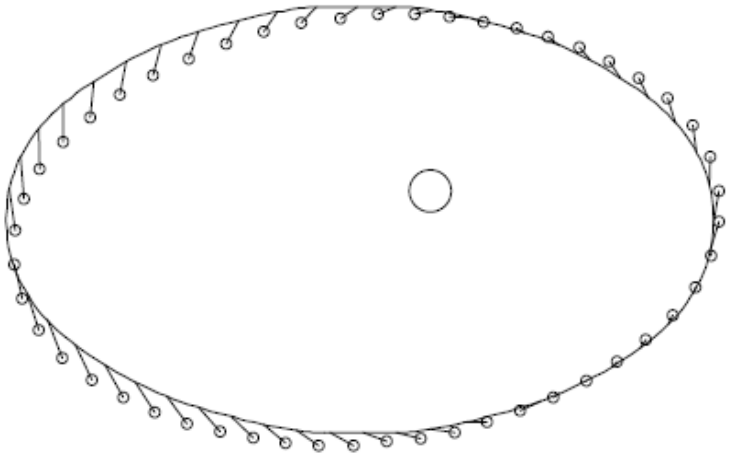
El punto más importante en éste algoritmo, es plantear una regla sensible para determinar correspondencia sin conocimiento alguno de rotación y traslación. En el estudio, proponen dos reglas, y las implementan juntas como solución óptima. La primera de las reglas, denominada regla del punto más cercano, toma la decisión en base a su nombre. Busca el punto más cercano a P_i en el mapa S_{ref}, y los asigna como pareja. En la Figura No. 160 se puede observar un ejemplo de aplicación de la regla. El problema con ésta regla, de acuerdo a los autores, es la lentitud con la que converge, y la poca información que da sobre la rotación. [29]

FIGURA NO. 160. APLICACIÓN DE LA REGLA DE PUNTO MÁS CERCANO. [29]



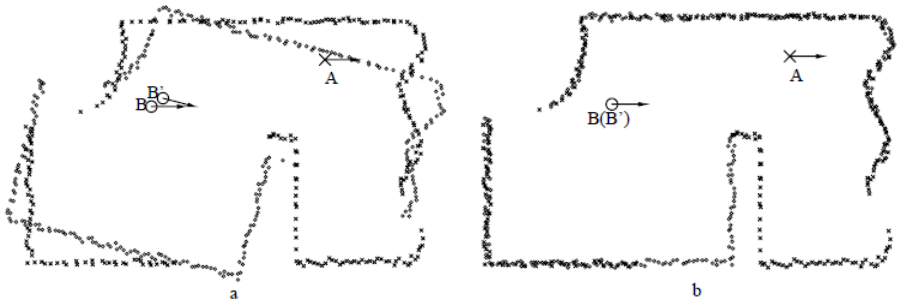
La segunda regla propuesta, llamada regla de coincidencia rango punto, busca proveer el dato de la rotación. La idea de la regla es buscar, en una región limitada alrededor de cada punto P_i, un punto P'_i con la misma distancia radial que $P_i$con respecto al origen. La región se va limitando más con cada iteración, hasta llegar a la convergencia. En la Figura No. 161 se puede observar un ejemplo de aplicación de la regla. [29]

FIGURA NO. 161. APLICACIÓN DE LA REGLA DE COINCIDENCIA RANGO PUNTO. [29]



En la investigación, proponen una combinación de ambas reglas. Aplican la primera para obtener la traslación, la segunda para la rotación, y se aplican los valores obtenidos para ajustar S_{ref} en cada iteración. En la Figura No. 162 se muestran resultados del estudio.

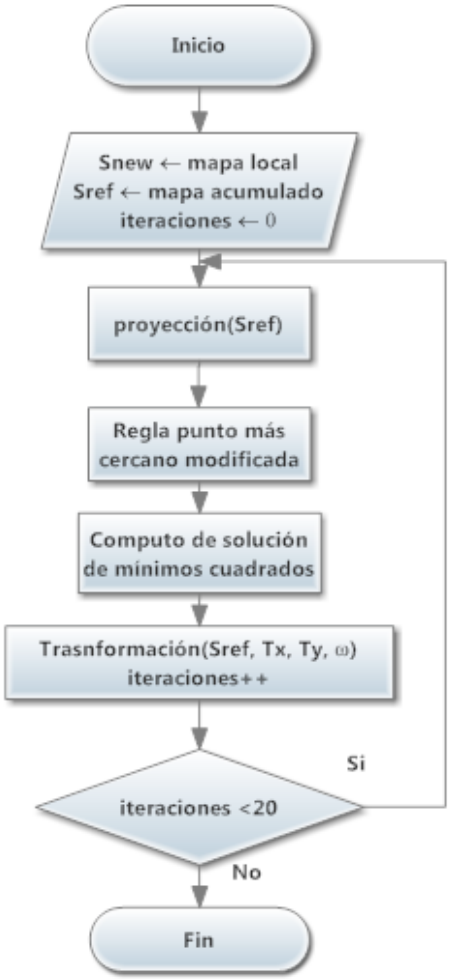
FIGURA NO. 162. APLICACIÓN DE ALGORITMO DE ALINEAMIENTO PROPUESTO. [29]



Con base a las ideas y conocimientos planteados en el estudio mencionado, se procede a detallar el algoritmo de alineamiento y coincidencia desarrollado en este trabajo.

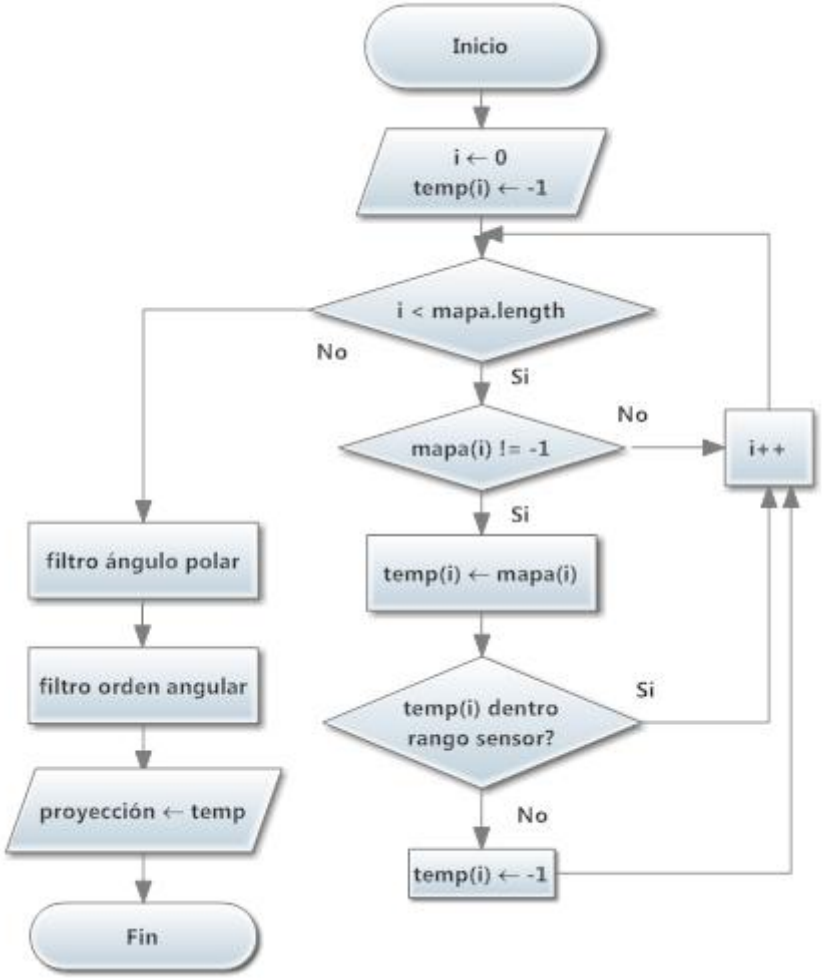
2) **Planteamiento de algoritmo de alineamiento y coincidencia.** El algoritmo desarrollado se muestra en el diagrama de flujo de la Figura No. 163:

FIGURA NO. 163. DIAGRAMA DE FLUJO DE ALGORITMO DE ALINEAMIENTO Y COINCIDENCIA



En el diagrama, se puede observar que las entradas más importantes del algoritmo son el mapa local nuevo, el mapa acumulado anterior trasladado y una cuenta para mantener control del número de iteraciones. El primer bloque de procesamiento, corresponde al algoritmo de proyección. Este algoritmo implica múltiples etapas, por lo que también se detalla su diagrama de flujo en la Figura No. 164.

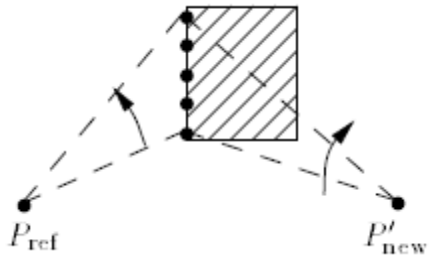
FIGURA NO. 164. DIAGRAMA DE FLUJO ALGORITMO DE PROYECCIÓN



Es importante notar que el algoritmo de proyección, se aplica en dos ciclos o fases. En cada fase se evalúa todos los puntos del mapa acumulado trasladado. La primera etapa de la proyección, consiste en filtrar todos los puntos dentro del alcance del sensor de manera individual. La primera validación es revisar si el punto a trabajar es válido. Si esto se cumple, se guarda temporalmente las coordenadas del punto, tanto en coordenadas cartesianas como en polares. Con esta información, se verifica si las coordenadas polares del punto se encuentran dentro del rango del sensor. Es decir, si la distancia radial es menor a 4096 y si el ángulo se encuentra entre 0º y 240º. Todos los puntos que no pasan el primer filtro, toman el valor de punto inválido (-1) en el arreglo temporal que se va a procesar en la segunda etapa.

Con el arreglo temporal establecido, se busca aplicar un segundo filtro para eliminar puntos que no se deben de considerar en el alineamiento. Estos puntos que no se deben de considerar, se filtran por dos casos. El primer caso, se da cuando dos o más puntos tienen el mismo ángulo polar. Si esto sucede, se deben de eliminar los puntos más lejanos. El segundo caso tiene un grado de complejidad más alto. En este caso se revisan conjuntos de puntos consecutivos, en coordenadas cartesianas. Se van recorriendo los puntos en sentido contrario a las agujas del reloj (sentido en que el sensor los genera), y se va verificando la coordenada angular. Para que los puntos sean válidos, el ángulo debe de ir incrementando. Cuando el ángulo empieza a decrecer, quiere decir que es una superficie no visible para el sensor y se debe de eliminar. Este segundo caso se muestra en la Figura No. 165. Se puede notar que para el mapa acumulado sin traslación (P_{ref}), los puntos van incrementando su coordenada angular. Cuando se cambia el punto de referencia (cambio de coordenadas), estos puntos ya no serían visibles para el robot, y su ángulo va en decremento al recorrer el mapa en el mismo sentido.

FIGURA NO. 165. SUPERFICIE NO VISIBLE PARA PROYECCIÓN



Al finalizar el bloque de proyección, se cuenta con dos mapas con la misma cantidad de puntos. El mapa local nuevo, y el mapa proyectado. Estos dos mapas son la entrada para el segundo bloque de procesamiento, la regla del punto más cercano modificada. Como se mencionó con anterioridad, esta regla se basa en la regla de punto más cercano planteada en el estudio [29], pero se le aplicaron ciertas modificaciones para mejorar la regla, por lo que se presenta como parte de éste trabajo.

La regla original, simplemente busca emparejar todos los puntos del mapa local nuevo con su punto más cercano en el mapa acumulado anterior trasladado. Esta regla resulta ineficiente, debido a que toma en consideración todas las parejas de puntos

posibles. No considera la existencia de puntos nuevos en el mapa local que no tienen un correspondiente en el mapa acumulado anterior. Estas parejas de puntos nuevos, se dejan fuera del posterior análisis en la regla elaborada en éste trabajo. El criterio de omisión de parejas, se basa en plantear una región angular válida para encontrar el punto más cercano. Si la pareja asignada se encuentra fuera de esta región, se deja de considerar estos puntos como pareja.

La segunda modificación a la regla, busca mejorar la calidad del algoritmo de alineamiento filtrando de una manera estricta las parejas de puntos correspondientes. En la regla original, puede darse el caso en que dos puntos del mapa local nuevo, tengan al mismo punto en el mapa acumulado trasladado como su pareja. Es decir la relación entre puntos no es biunívoca. La modificación implementada es revisar cada pareja de puntos, y cuando se encuentren casos de puntos con la misma pareja, se evalúa cuál de las dos parejas tiene una distancia menor. Se elimina la pareja con la distancia mayor.

Finalmente, la última modificación pretende hacer un último filtrado de parejas de puntos no relevantes. Este último criterio implementado revisa si la distancia entre la pareja de puntos es mayor a 1/10 del alcance máximo del sensor, se elimina la pareja. Esto se hace con el fin de solo dejar como parte del análisis las parejas de puntos más cercanas.

Aplicando estas tres modificaciones, se considera que la regla de punto más cercano mejora su calidad de una manera significativa.

El tercer bloque de procesamiento, corresponde al cómputo de la solución de mínimos cuadrados, con el fin de obtener los valores de traslación y rotación que minimicen la función de distancia. Este bloque es bastante sencillo, pues simplemente recibe como entrada las parejas de puntos correspondientes, y les aplica la forma cerrada de la función de distancia desarrollada en el apéndice A. Se obtiene del bloque los valores de traslación y rotación que se deben de aplicar al mapa acumulado anterior para disminuir el error.

El cuarto y último bloque, corresponde a la aplicación de las correcciones de traslación y rotación obtenidas en el bloque anterior, sobre el mapa acumulado anterior.

Este proceso va modificando el mapa en sus coordenadas cartesianas y polares simultáneamente. La primera transformación que se aplica es la rotación, y ésta se aplica sobre el mapa en coordenadas polares. A cada punto en el mapa se le añade la corrección ω en su coordenada angular, y simultáneamente se recalcula la coordenada cartesiana del mismo. Al finalizar el ajuste angular, se aplica la transformación de traslación. Esta se aplica sobre cada punto en el mapa en coordenadas cartesianas, y simultáneamente se va recalculando la coordenada polar del mismo.

Al finalizar la aplicación de los cuatro bloques, se logra obtener un nuevo mapa acumulado anterior con menos error con respecto al nuevo mapa local. Este proceso se repite durante la cantidad de iteraciones necesaria para dar buenos resultados, tomando en cuenta que las entradas para la siguiente iteración son el mapa local nuevo, y el mapa acumulado anterior corregido.

b. Resultados

FIGURA NO. 166. RESULTADO 1 ALGORITMO DE PROYECCIÓN

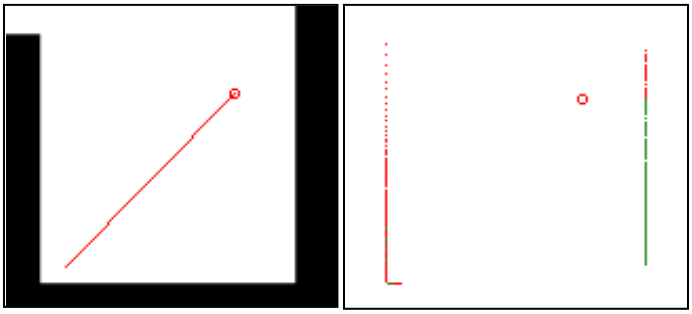


FIGURA NO. 167. RESULTADO 2 ALGORITMO DE PROYECCIÓN

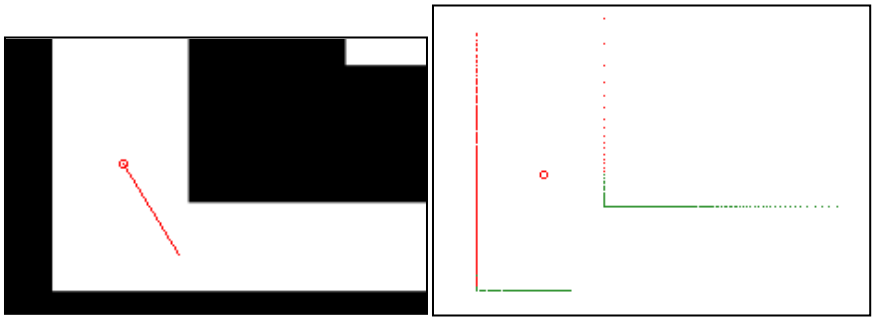


FIGURA NO. 168. RESULTADO 3 ALGORITMO DE PROYECCIÓN

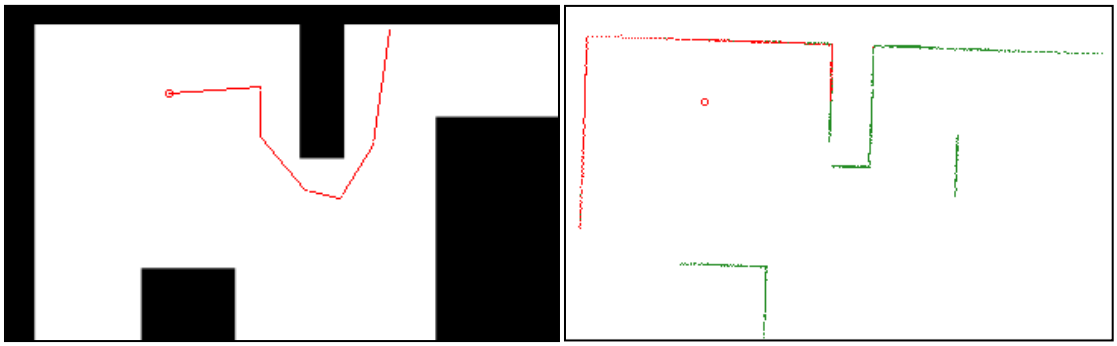


FIGURA NO. 169. RESULTADO CON 1 ITERACIÓN ALGORITMO DE ALINEAMIENTO

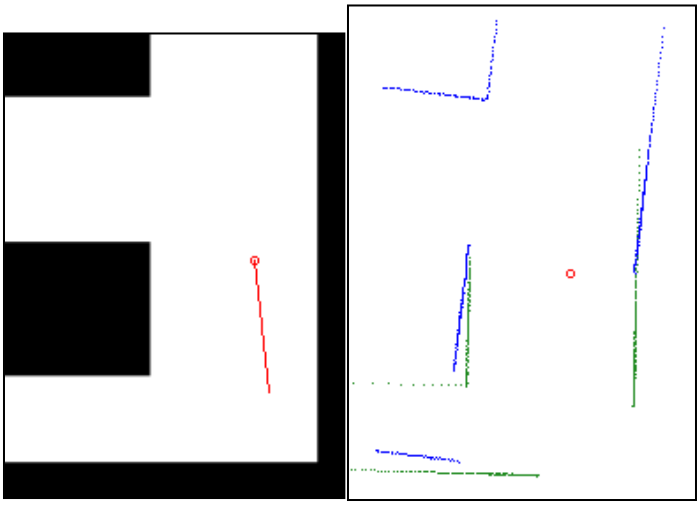


FIGURA NO. 170. RESULTADO CON 5 ITERACIONES ALGORITMO DE ALINEAMIENTO

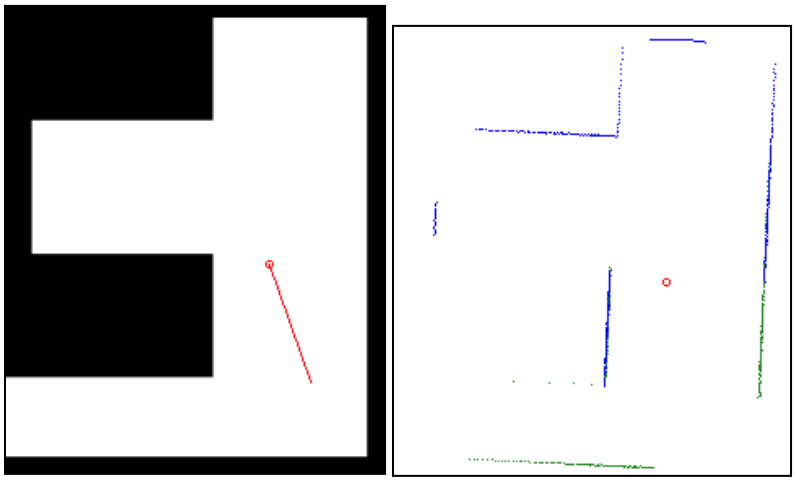


FIGURA NO. 171. RESULTADO CON 15 ITERACIONES ALGORITMO DE ALINEAMIENTO

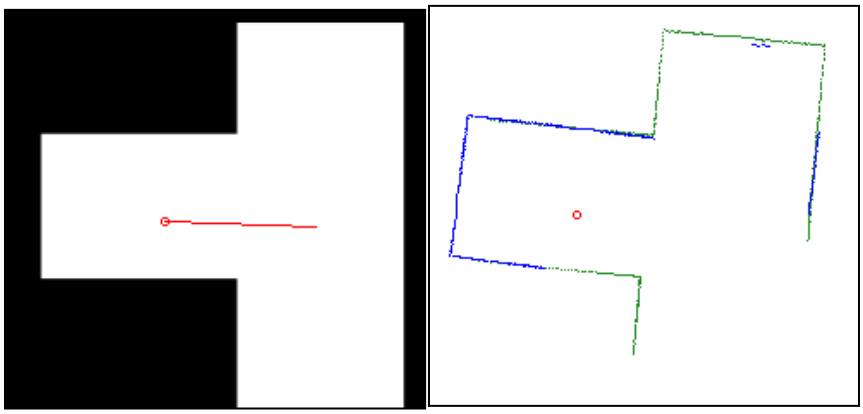


FIGURA NO. 172. RESULTADO CON 40 ITERACIONES ALGORITMO DE ALINEAMIENTO

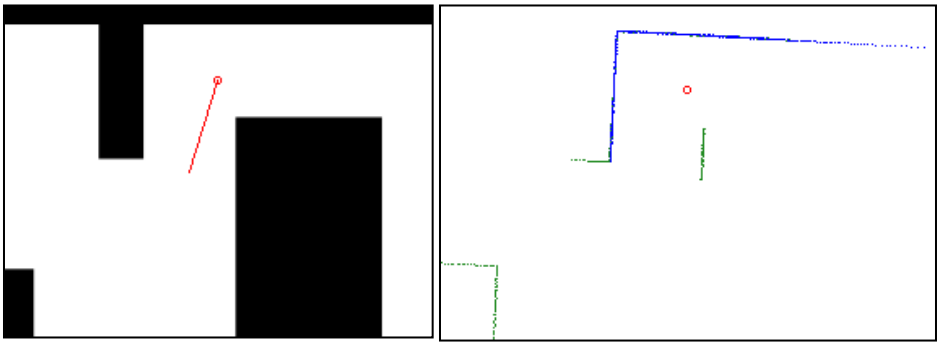


FIGURA NO. 173. SECUENCIA DE MOVIMIENTOS PARA ANÁLISIS DE CONVERGENCIA

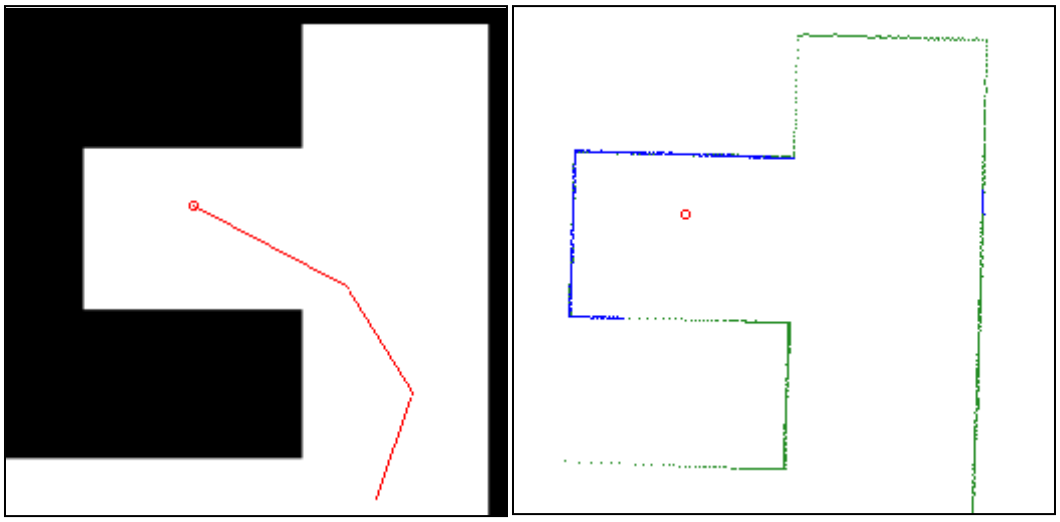


FIGURA NO. 174. RESIDUALES DE ROTACIÓN MOVIMIENTO 1

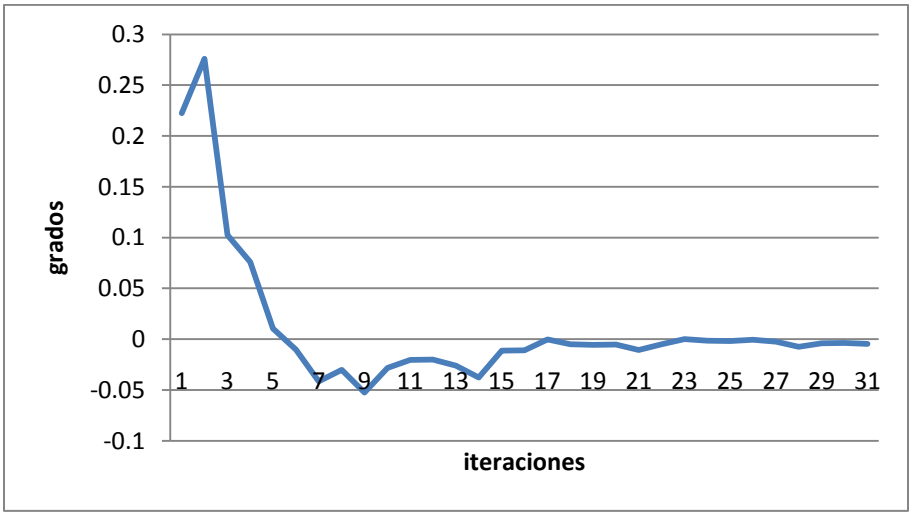


FIGURA NO. 175. RESIDUALES DE TRASLACIÓN MOVIMIENTO 1

FIGURA NO. 176. RESIDUALES DE ROTACIÓN MOVIMIENTO 2

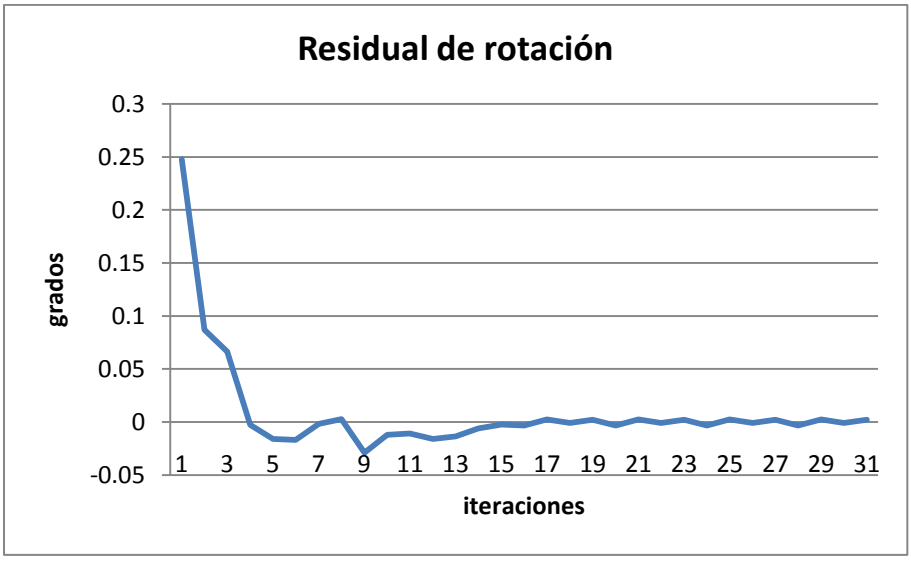


FIGURA NO. 177. RESIDUALES DE TRASLACIÓN MOVIMIENTO 2

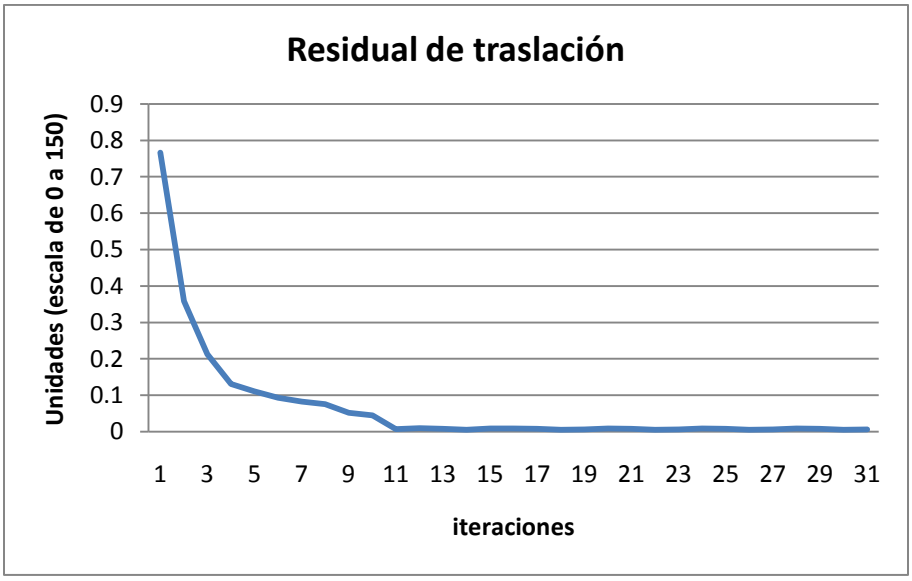


FIGURA NO. 178. RESIDUALES DE ROTACIÓN MOVIMIENTO 3

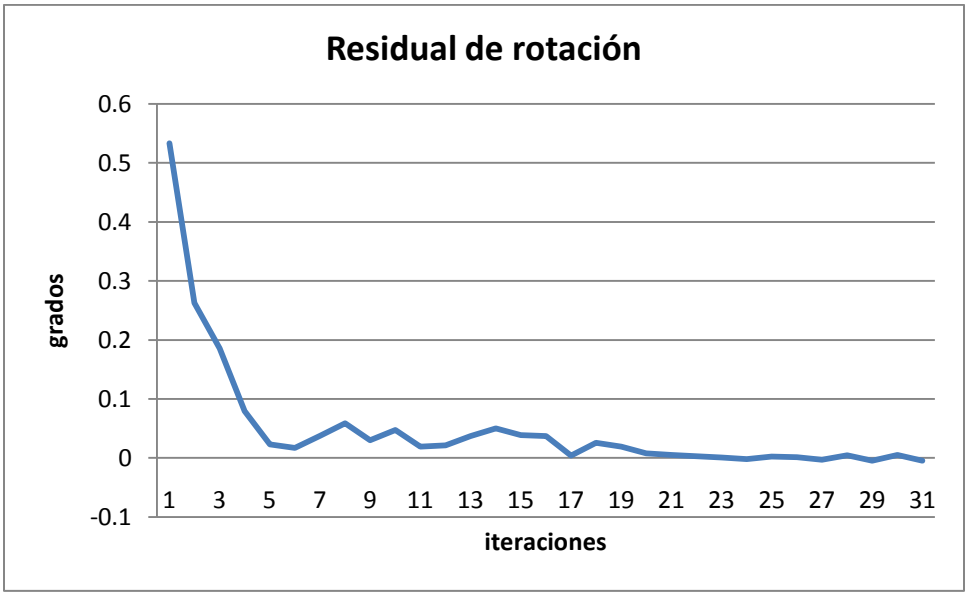
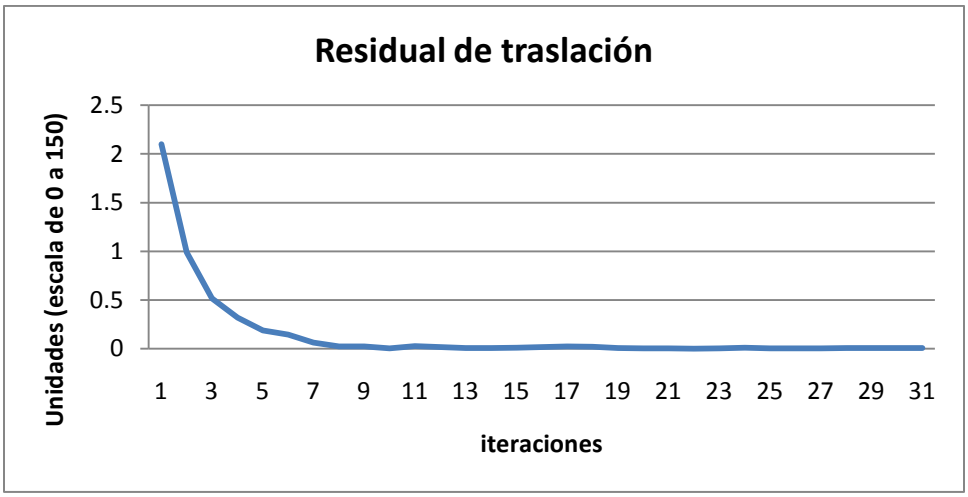


FIGURA NO. 179. RESIDUALES DE TRASLACIÓN MOVIMIENTO 3



c. ***Discusión.*** En la Figura No. 166, Figura No. 167 y Figura No. 168, se muestran los resultados de la aplicación del algoritmo interno de proyección. Se buscó mostrar con estos tres casos, la solidez del algoritmo, al ponerlo a prueba en situaciones que podrían causarle conflicto. En la Figura No. 166, se muestra una proyección simple sobre un mapa pequeño. Se puede observar que la proyección (puntos rojos) está limitada por el rango del sensor. Es importante mencionar que el origen o 0 del sensor, está ubicado a la derecha de la ubicación del robot, correspondiendo exactamente con el 0 del plano polar. Existen segmentos del mapa acumulado (puntos verdes) que se omiten en la proyección, por estar fuera del rango del sensor. En la Figura No. 167 se muestra un caso similar pero con un grado más alto de complejidad.

En la Figura No. 168, se muestra uno de los casos más complejos a los que se puede enfrentar el algoritmo de proyección. En este resultado se muestra un mapa acumulado más completo que los dos anteriores, y se construye alrededor de una pared en la simulación que podría causar conflicto al algoritmo. La dificultad está en el hecho que dentro del rango del sensor, se encuentran puntos que corresponden al lado de la pared que no es visible para el robot. El algoritmo logra de manera exitosa omitir estos puntos con la regla de omisión por incremento de ángulo, y solo considera puntos visibles desde la posición del robot.

En la Figura No. 169, Figura No. 170, Figura No. 171 y Figura No. 172, se muestran resultados de la aplicación del algoritmo de alineamiento completo, con distintos números de iteraciones para el algoritmo. La cantidad de iteraciones entre estos resultados va en aumento, y se puede notar como con más iteraciones mejora la calidad del alineamiento, sin embargo a partir de la Figura No. 171, la calidad de los resultados parece haber alcanzado un máximo.

La cantidad de iteraciones necesarias para asegurar un buen alineamiento, fue un valor experimental que se estableció a partir de distintos análisis de pruebas de simulación. Uno de estos análisis, se incluye en los resultados para reafirmar la validez del valor establecido. En la Figura No. 173, se muestra una trayectoria del robot simulado compuesta por tres movimientos. El mapa obtenido de esta trayectoria, se

muestra junto a la misma. Se puede notar la calidad del alineamiento al compararlo con el ambiente simulado.

Para cada movimiento, se aplicaron 30 iteraciones del algoritmo de alineamiento. Para cada iteración, se recolectó el valor de ajuste en rotación y traslación que se aplica al mapa acumulado para corregirlo. Estos valores, se denominan residuales debido a que corresponden a la cantidad restante por ajustar para lograr un alineamiento exacto, de acuerdo con el análisis del algoritmo. Para obtener el residual de traslación total, se calculó la traslación resultante de las componentes de traslación en x y en y por medio del teorema de Pitágoras. Estos residuales se grafican vs el número de iteración correspondiente a cada uno en la Figura No. 174, Figura No. 175, Figura No. 176, Figura No. 177, Figura No. 178 y Figura No. 179. Se puede observar que existe una tendencia común en los tres movimientos, como también se notó en todos los demás experimentos realizados. Se puede notar que las primeras 5 iteraciones, son las que más contribuyen a eliminar el error entre mapas, tanto en el parámetro de rotación como en el de traslación. En la mayoría de los casos, la primera iteración es la más fuerte, y luego va decreciendo la magnitud del residual. En otros casos, la segunda o tercera iteración da el residual más grande, y luego decrece su magnitud en las siguientes iteraciones. A pesar de que los residuales decrecen drásticamente hasta la 5ta iteración, estos aún no son cero, por lo que éste número no es suficiente para un alineamiento exacto.

Se puede observar que el comportamiento más estable se da en los residuales de traslación, e incluso se podría considerar una función inversa para modelar su comportamiento. Si la cantidad de iteraciones dependiera únicamente del parámetro de traslación, se hubiera considerado como un valor aceptable 15. Sin embargo, el comportamiento de los residuales de rotación no es tan controlado como el de traslación. Incluso se dan casos en que el residual cambia de signo y se da cierto tipo de oscilación. Este comportamiento se observa claramente en la Figura No. 174 y Figura No. 176. Sin embargo, a pesar de las oscilaciones, el residual de rotación converge de igual forma que el de traslación a un residual de magnitud cero. La cantidad de iteraciones necesarias para llegar a esta convergencia, se puede establecer en 20. Esta cantidad de iteraciones resulta un valor óptimo, pues garantiza buenos resultados por parte del algoritmo de

alineamiento, y mantiene el tiempo de ejecución bajo, evitando iteraciones de más que serían innecesarias.

Un aspecto notable, es que la convergencia de ambos residuales a cero, reafirma la calidad de la regla de coincidencia de puntos modificada propuesta en este trabajo, pues no solo da buenos resultados para la traslación, sino también para la rotación. Esta regla de coincidencia única, presenta ventajas con respecto a la regla combinada presentada en [29], pues en tiempo, tiene un costo más bajo, y este parámetro viene a ser de gran importancia en el megaproyecto SEEQ, como se discutirá posteriormente.

3. ALGORITMO DE UNIÓN DE MAPAS Y OMISIÓN DE REPETICIONES

a. Diseño Experimental. Ya alineados y con el mismo punto de referencia el mapa local nuevo y el mapa acumulado anterior, solo queda un paso por implementar para lograr la construcción progresiva de mapas. Este paso consiste en unir estos dos mapas para dar lugar a un mapa acumulado nuevo, o mapa global como se le llamará en adelante.

En las etapas iniciales de desarrollo, se consideró como solución un algoritmo de unión muy simple, que toma todos los puntos de ambos arreglos (tanto en coordenadas cartesianas como en polares), y los almacena en un nuevo arreglo, el cual es llamado mapa global. Sin embargo, esta solución comenzó a presentar sus desventajas con las pruebas de trabajo en la simulación. La principal desventaja de este algoritmo, es el rápido crecimiento de cantidad de puntos, y por lo mismo el tiempo de procesamiento aumenta de manera significativa. El problema es que aunque no se esté detectando partes nuevas del ambiente, se están agregando siempre todos los puntos del mapa local nuevo, incluyendo puntos repetidos.

Se propone entonces un algoritmo un poco más elaborado, que busca eliminar puntos repetidos en el mapa global, de manera que los atributos en el ambiente sean representados por la menor cantidad posible de puntos.

El algoritmo de unión de mapas y omisión de puntos repetidos, se basa en el algoritmo de alineamiento presentado en la sección anterior. La idea es que una vez realizadas las 20 iteraciones del algoritmo de alineamiento, se aplique una vez más la regla de correspondencia de puntos modificada, con el fin de obtener un listado de parejas de puntos correspondientes. Estos puntos correspondientes, se consideran como repeticiones, pues ambos puntos están describiendo el mismo lugar en el ambiente. El algoritmo entonces toma como entrada este listado de puntos correspondientes entre el mapa local nuevo y el mapa de la proyección del mapa acumulado anterior, y elimina los puntos de la proyección que estén en el listado. Estos puntos son eliminados directamente en el mapa acumulado anterior, pues es este el que se va a unir con el mapa local nuevo.

El siguiente paso en el algoritmo es incorporar los puntos del mapa local nuevo, y recalcular las coordenadas cartesianas del mapa completo. Al terminar el algoritmo, se cuenta entonces con un mapa global en coordenadas polares y otro en coordenadas cartesianas.

b. Resultados

FIGURA NO. 180. MAPA GLOBAL UNIÓN SIN ELIMINACIÓN DE REPETIDOS

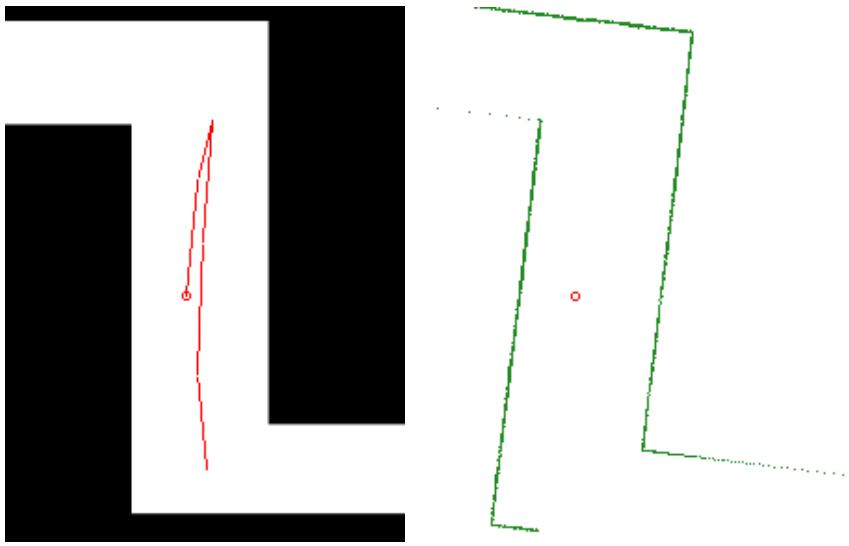


FIGURA NO. 181. MAPA GLOBAL UNIÓN CON ELIMINACIÓN DE REPETIDOS

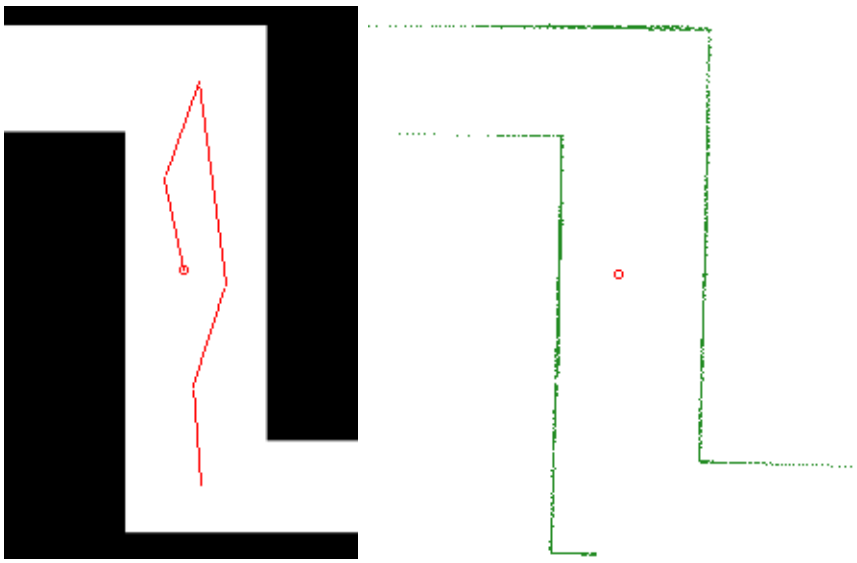


TABLA NO. 8. COMPARACIÓN DE TIEMPOS DE EJECUCIÓN POR TÉCNICA DE UNIÓN

Caso	**Tiempo (m: s: ms)**
Unión sin eliminación de repetidos	2:13:72
Unión con eliminación de repetidos	1:16:14

c. Discusión. En la Figura No. 180 y Figura No. 181, se muestran los dos casos del algoritmo de unión trabajado. Para el experimento, se aplicó una trayectoria similar a ambos algoritmos, y se midió el tiempo de ejecución de cada trayectoria. La trayectoria constó de 5 movimientos en el ambiente simulado 2, y se obtuvo una representación visual de cada mapa global generado para poder hacer una comparación visual y por tiempos.

Cuando se aplicó el algoritmo de unión simple, sin eliminación de repeticiones, el mapa obtenido tiene una cantidad de puntos muy extensa, con un total de 3415 puntos. Estos puntos son en su mayoría repeticiones del mismo aspecto del ambiente, y es por esto que en la representación visual del mapa, se ve una gran densidad de puntos en las paredes que detectó el robot más de una vez.

En cambio, aplicando el algoritmo de unión con eliminación de repeticiones, el tamaño del mapa obtenido se redujo a 1625 puntos, y esta diferencia se ve claramente al comparar la densidad de puntos en la Figura No. 180 con la Figura No. 181.

En términos de tiempos, el algoritmo sin eliminación tardó 2 minutos y 13 segundos en ejecutar la trayectoria, mientras que el algoritmo con eliminación lo realizó en 1 minuto y 16 segundos. Es decir se redujo a casi la mitad el tiempo.

Estos resultados nos indican que si es necesario limpiar el mapa global de repeticiones, para poder tener un mapeo eficiente. Se podría pensar que para 5 movimientos, el tiempo de ejecución de 1 minuto y 16 segundos es aún bastante lento, pero esto se atribuye en parte a la actualización de la interfaz gráfica del ambiente de simulación. Este tiempo se va reducir en la implantación real discutida posteriormente.

C. ALGORITMOS PARA EXPLORACIÓN

En esta sección, se desarrolla la lógica del algoritmo de exploración implementado en el módulo de vuelo automatizado. El algoritmo de exploración presentado y utilizado en éste trabajo, no fue un diseño ni desarrollo propio, sino fue más bien una técnica o teoría propuesta por investigadores de otro país, que obtuvieron buenos resultados. El motivo por el cual no se elaboró un algoritmo de exploración propio, es porque los conocimientos detrás de un algoritmo de exploración robusto, basado en análisis estadísticos y no en simple prueba y error, sobrepasan el nivel de conocimientos de un grado de licenciatura.

El trabajo en este algoritmo consistió entonces, en la comprensión de la teoría que lo formula, además de la implementación práctica de esta teoría la cual no es presentada por sus autores.

La función de un algoritmo de exploración, es darle a un robot la capacidad de tomar la decisión de su siguiente movimiento, con el fin de aumentar el conocimiento del ambiente. Es decir busca planificar movimientos para expandir al máximo el mapa

construido. El algoritmo de exploración implementado, denominado «Estrategia de información basado en Información» [2] por sus autores, establece las mejores posiciones de observación tomando en cuenta la distancia al punto, y la información recopilada. La planificación se basa en el análisis de la posición actual y el mapa global construido hasta el momento, para obtener los parámetros de decisión. La estrategia presentada está fundamentada por un análisis matemático robusto, basado en el concepto de entropía relativa.

Como una previa al algoritmo de exploración, se presentan otras técnicas de exploración utilizadas durante la etapa de desarrollo, en el ambiente simulado. Esto con el fin de mostrar las ventajas que presenta el algoritmo final sobre ellas.

1. TÉCNICAS DE EXPLORACIÓN EN AMBIENTE SIMULADO

a. Diseño experimental. La técnica más simple, y menos eficiente que se trabajó durante el desarrollo del proyecto, se basa en las funciones de generación de números aleatorios incorporadas en cualquier lenguaje de programación.

La estrategia consiste en generar un número aleatorio para la coordenada en x y la coordenada en y, alrededor de la posición actual del robot. Este punto se valida con las reglas establecidas en el capítulo de simulación, y si no es válido, se vuelve a generar otra opción hasta encontrar un punto válido.

La segunda técnica utilizada, fue permitirle al usuario trazar la ruta en el ambiente simulado, por medio de la selección de la siguiente posición a partir del último movimiento. Esta técnica se implementó para poder elaborar mapas completos de una manera eficiente que con la exploración aleatoria.

b. Resultados

FIGURA NO. 182. EXPLORACIÓN ALEATORIA - 6 MOVIMIENTOS Y SU MAPA GENERADO

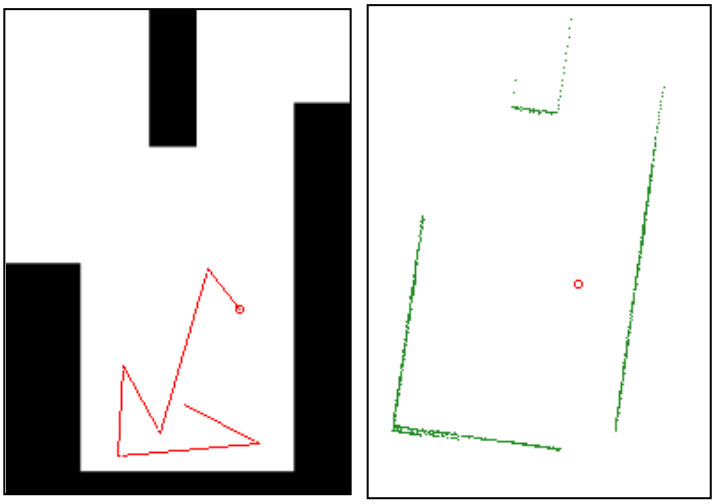


FIGURA NO. 183. EXPLORACIÓN TRAZADA POR USUARIO - 6 MOVIMIENTOS

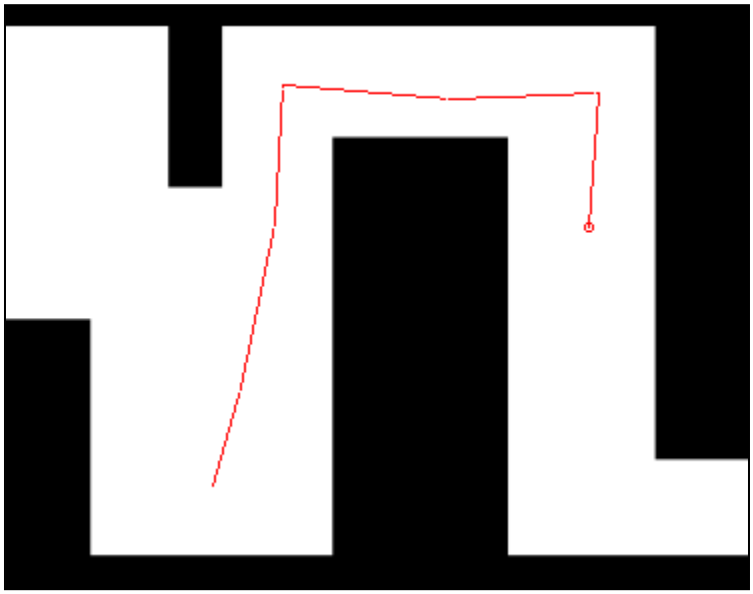
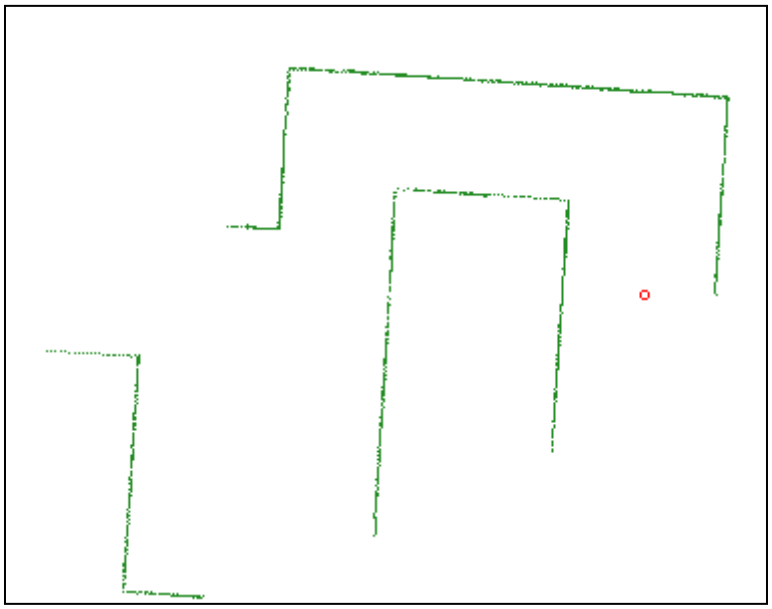


FIGURA NO. 184. MAPA GENERADO POR EXPLORACIÓN DIRIGIDA POR USUARIO



c. Discusión. En la Figura No. 182, se puede observar el recorrido y el mapa creado con exploración aleatoria en 6 movimientos. Se puede observar que el recorrido no es óptimo, pues el robot se mueve a puntos en los que no obtiene información nueva para el mapa, y no se tiene una lógica de planeación. Esta técnica no utiliza nada de información sobre su ambiente más que los puntos en los que no puede ubicarse el robot.

La técnica de trazo de ruta por el usuario, no se considera como opción para el resultado final del proyecto, pues se perdería la autonomía del robot, dependiendo del usuario para toma decisiones. Sin embargo, se puede apreciar los resultados de los algoritmos de construcción de mapas utilizándola. En la Figura No. 183 y Figura No. 184 se muestra su aplicación.

2. EXPLORACIÓN BASADA EN INFORMACIÓN

a. Diseño Experimental. El criterio propuesto para construir mapas de manera eficiente, toma en cuenta dos aspectos esenciales. De ellos obtiene la información necesaria para tomar decisiones y proponer la siguiente mejor posición de exploración. Los criterios son:

- La distancia recorrida para llegar al punto de observación
- La información que se espera haber recopilado

El primer criterio, corresponde a la distancia viajada para alcanzar la posición objetivo. Este primer criterio, considera el costo C de moverse en el ambiente para poder mapearlo completamente. Es necesario darle al algoritmo, un valor teórico del perímetro del ambiente a explorar, para definir el criterio en términos de C de la siguiente manera: [2]

$$E[C] = P - a$$

La expresión anterior indica que la distancia a recorrer por el robot para construir el mapa del ambiente con perímetro P está en términos de la diferencia entre el perímetro y el alcance del sensor definido como un perímetro a. Si el área abarcada por el sensor es muy grande, se cumple $P \approx a$, y el criterio establecería $E[C] = 0$. Esto nos indicaría que el robot no necesita moverse para mapear el ambiente. En el caso contrario, en que el área del sensor es muy pequeña con respecto al perímetro del ambiente, el criterio establecería $E[C] = P$, indicándonos que el robot tendría que recorrer una distancia igual al perímetro del ambiente para formar un mapa representativo. [2]

Se puede adaptar este criterio para determinar el número de movimientos requerido por el robot. Asumiendo que en cada movimiento se explora un segmento igual al del escaneo a, y que el perímetro del ambiente es significativamente mayor que a, es necesario realizar n movimientos, en donde $n = \frac{P}{a}$.

El segundo criterio, corresponde a la cantidad de información que se espera obtener con el movimiento realizado. Este parámetro de información, se mide a través de la incertidumbre del mapa al agregar la información del nuevo punto de observación.

La incertidumbre en el punto de medición, se ve afectada por dos componentes. El primero es la incertidumbre de la medición del sensor, que en el caso del sensor utilizado en el proyecto tiene un valor aproximado de 30 mm.

El segundo componente es la incertidumbre de la posición del robot en el punto de observación. Se debe de tener una medición estimada de cuanto se desplaza el robot para llegar al punto de observación. Esta medición se ve afectada por errores dependientes del tipo de trayectoria seguida. [2]

El criterio propuesto en el estudio se basa en el concepto de la entropía relativa. Este concepto se desarrolla de manera completa en el apéndice de mapeo y exploración, y solo se presenta aquí el resultado final. [2]

FIGURA NO. 185. CRITERIO PROPUESTO

$$E[\Delta H] \approx \frac{1}{N+A} \sum_{i \in \mathcal{A} \cup \mathcal{N}} ln \frac{\sigma_{unc,i}}{\sigma} + N\, ln \frac{\sigma}{P} + \sum_{i \in \mathcal{A}} ln \frac{\sigma}{\sigma_{p,i}}$$

Lo que nos indica la expresión, es que la incertidumbre en una nueva posición, está dada por tres términos. El primer término, corresponde a la incertidumbre producida por el error de posición. El segundo, a la incertidumbre de los nuevos puntos obtenidos, y el tercero a los puntos previamente obtenidos, es decir el mapa acumulado. Al alcanzar el robot una nueva posición de observación, la incertidumbre se enfrenta a dos tendencias. Por un lado, se espera que la incertidumbre aumente por el error de posición, pero si el robot visita localidades con mapa previamente creado, se disminuye la incertidumbre. [2]

Habiendo detallado cada criterio del algoritmo, se procede a combinarlos para presentar el criterio final. Tomando en cuenta que el primer criterio establece que el número de movimientos esperados es $n = \frac{P}{a}$, se puede decir que $P = na$. Se define entonces la cantidad de información que se podría obtener de una nueva posición como: [2]

FIGURA NO. 186. CANTIDAD DE INFORMACIÓN QUE SE PODRÍA OBTENER DE UNA NUEVA POSICIÓN

$$\overline{\Delta H} \approx -N\, ln\frac{a}{\sigma} = -N\, ln\frac{4\pi\, c}{3\sigma}$$

En donde c corresponde al alcance máximo del sensor, que al mismo tiempo debería de ser la distancia que se mueve el robot para su siguiente posición, que en el caso de este trabajo es 4095 mm. El término $\frac{4\pi}{3}$ corresponde a los 240º de rango del sensor.

Entonces, se formula el criterio de selección como la diferencia entre las incertidumbres del segundo criterio, y la cantidad de información nueva que se podría obtener en la nueva posición. Esto se especifica como:

FIGURA NO. 187. CRITERIO DE SELECCIÓN J

$$J = E[\Delta H] - \overline{\Delta H} = E[\Delta H] + N\, ln\frac{4\pi\, c}{3\sigma} = \frac{1}{N+A}\sum_{i\in\mathcal{A}\cup\mathcal{N}} ln\frac{\sigma_{unc,i}}{\sigma} + N\, ln\frac{\sigma}{P} + \sum_{i\in\mathcal{A}} ln\frac{\sigma}{\sigma_{p,i}} + N\, ln\frac{4\pi\, c}{3\sigma}$$

La información que se espera obtener $\overline{\Delta H}$, representa el decremento de entropía asociado a un movimiento en el ambiente. Se puede decir que el criterio es entonces la diferencia entre la información que se espera obtener, y la información que podría obtenerse bajo condiciones ideales. Valores pequeños de J identifican las mejores posiciones de observación. [2]

Con el criterio formulado, el algoritmo de exploración se reduce a computar J para cada punto de observación candidato, y se elige como mejor posición el punto con menor valor obtenido. Es importante, para obtener buenos resultados, la selección de los candidatos para el siguiente movimiento.

En el algoritmo de exploración implementado en este proyecto, se creó un algoritmo de selección sencillo pero funcional. La idea es proponer como candidatos puntos a una distancia radial equivalente a la mitad del alcance máximo del sensor. Es decir, a 2048 mm. Los candidatos están distribuidos de manera uniforme en la región que abarca el sensor (240º), posicionando un candidato nuevo cada 15º. Esto nos da como resultado un total de 17 candidatos.

A cada candidato, se le computa el criterio de exploración, a través del algoritmo descrito a continuación.

Primero, se traslada el mapa global construido hasta el momento, al punto candidato como referencia. Se aplica el algoritmo de proyección, y este mapa obtenido se almacena como A. Luego se calcula $\mathcal{A}$, el cual corresponde a la cantidad de puntos válidos en A. Del mismo valor computado, se puede obtener el valor de $N = 683 - A$. Este número corresponde al número de puntos nuevos que se espera escanear en la nueva posición. Después de obtener estos valores, se le da valor al término $\sigma_{unc\,,i}$ del criterio. El valor de este está en términos del error esperado en rotación, el error esperado en traslación, y la distancia entre el candidato y la posición actual. El error en rotación se establece como 0.01, el error en traslación como 0.001, y la distancia se limita a un máximo de 2048 mm. Posteriormente se asigna un valor a σ, $\sigma_{p,i}$y ya con todos los valores establecidos, se calcula J.

Después de calcular el criterio para un punto de observación, se retorna el mapa a la posición inicial, y se inicia el análisis del siguiente punto.

Existen variables en el criterio que son comunes para cualquier punto de observación. Estas variables y sus valores se listan a continuación:

TABLA NO. 9. VALORES DETERMINADOS PARA CRITERIO DE EXPLORACIÓN

Término	Descripción	Valor
V_θ	Error en rotación esperado	0.01
V_{xy}	Error en traslación esperado	0.001
c	Distancias máxima a viajar	2048 mm
$\sigma_{unc\,,i}$ simulación	$\sigma_{unc\,,i} = V_\theta c^2 + V_{xy}$	56.25
$\sigma_{unc\,,i}$ experimental	$\sigma_{unc\,,i} = V_\theta c^2 + V_{xy}$	41,943.04
σ	Incertidumbre de sensor	0.0001
P simulación	Perímetro	3000
P experimentación	Perímetro	100000

Al finalizar la selección del mejor punto de observación, se ejecuta el movimiento, y se cierra un ciclo de la lógica de mapeo y exploración.

b. Resultados

FIGURA NO. 188. EJEMPLO 1 -CRITERIO DE SELECCIÓN DE SIGUIENTE MOVIMIENTO

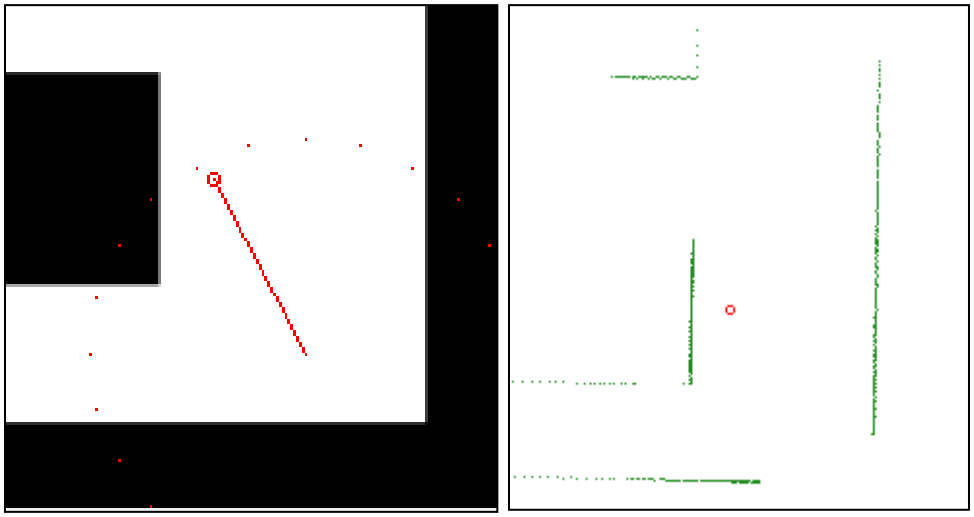


TABLA NO. 10. EJEMPLO 1 - RESULTADO DE CRITERIO DE SELECCIÓN POR CANDIDATO

No. Candidato	X	Y	Criterio J
1	37.50	64.95	-3099.2251
2	19.44	72.44	-3204.1356
3	0.00	75.00	-3297.3864
4	-19.41	72.44	-3460.5691
5	-37.50	64.95	-3489.3615
6	-72.44	19.41	-3239.1049
7	-75.00	0.00	-2953.1636
8	-72.44	-19.41	-3157.5092

FIGURA NO. 189. EJEMPLO 2 -CRITERIO DE SELECCIÓN DE SIGUIENTE MOVIMIENTO

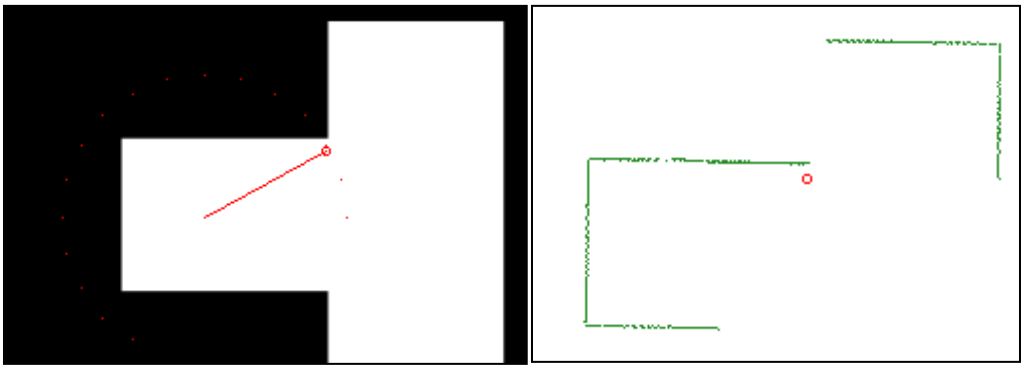


TABLA NO. 11. EJEMPLO 2 - RESULTADO DE CRITERIO DE SELECCIÓN POR CANDIDATO

No. Candidato	X	Y	Criterio J
1	75.00	0.00	-3017.6257
2	72.44	19.41	-3081.3932
3	64.95	37.50	-3186.3042

FIGURA NO. 190. EJEMPLO 3 - CRITERIO DE SELECCIÓN DE SIGUIENTE MOVIMIENTO

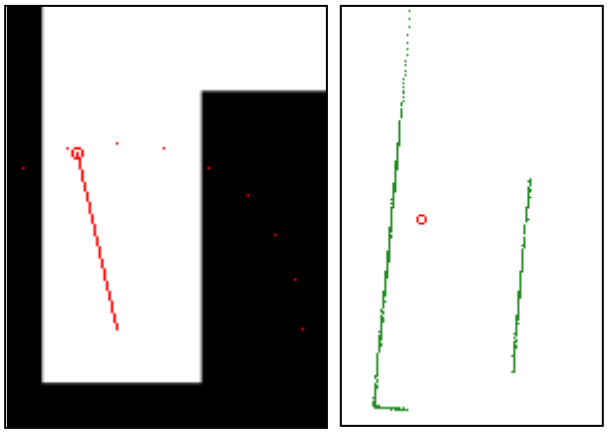


TABLA NO. 12. EJEMPLO 3 - RESULTADO DE CRITERIO DE SELECCIÓN POR CANDIDATO

No. Candidato	X	Y	Criterio J
1	19.41	72.44	-3483.8803
2	0.00	75.00	-3670.3645
3	-19.41	72.44	-3821.8763

c. Discusión. Se elaboraron diversas pruebas para verificar la calidad del algoritmo de exploración, y en la Figura No. 188, Figura No. 189 y Figura No. 190 se muestran los resultados de tres de estas pruebas.

Se puede observar que el algoritmo implementado, elimina como opciones todos los candidatos que atraviesan paredes u objetos. Esta eliminación es la primera fase de la lógica de selección de candidatos. Se realiza la eliminación antes de calcular el criterio de selección, para evitar procesamiento de datos innecesarios, y hacer más eficiente la ejecución. Se puede observar para cada caso de experimentación, que se obtuvieron las coordenadas (x, y) de cada candidato, y se muestra el resultado del computo del criterio de selección. Los puntos candidatos que se muestran en las tablas corresponden a los puntos válidos, recorriéndolos en sentido contrario a las agujas del reloj en la figura correspondiente.

Es bastante claro que el algoritmo es muy robusto, pues ante distintas situaciones, siempre elige la posición con mejores posibilidades de expandir el mapa construido.

El criterio implementado en el proyecto, difiere del propuesto por sus autores en [2], al igual que la técnica utilizada para evaluar los candidatos. En el estudio, los investigadores proponen generar una vista de 360 º para cada mapa local, haciendo rotar su sensor y aplicando el algoritmo de alineamiento a 4 escaneados para formar el mapa local completo. En cambio en el megaproyecto SEEQ, resulta muy complicado realizar esta rotación, ya que el robot es un vehículo aéreo. Además se considera innecesaria esta vista de 360º, pues con un sector de 240 º se adquiere suficiente información del ambiente para tomar decisiones.

Es por esta diferencia de mapas locales, que algunos parámetros en la ecuación del criterio difieren, pero sus resultados siguen siendo válidos y correctos.

D. IMPLEMENTACIÓN PRÁCTICA DE ALGORITMOS

Una vez desarrollados los algoritmos necesarios para llevar a cabo la tarea de exploración y creación de mapa, se procede a implementar la lógica completa, tanto en la simulación, como en el robot real.

La implementación en la simulación resultó bastante sencilla, pues simplemente consistió en enlazar todos los algoritmos trabajados.

Ahora para llevar a cabo la implementación real, se tuvieron que modificar algunos de los algoritmos para manejar la información de una manera ordenada y eficiente. La implementación real se llevó a cabo en distintas versiones debido a situaciones que exigieron distintos resultados del módulo. En esta sección se discutirán estas versiones y se mostrarán resultados de algunas de las mismas.

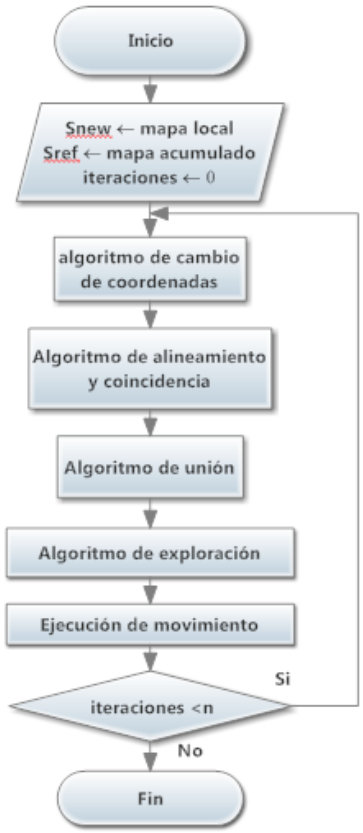
1. DISEÑO EXPERIMENTAL. En la etapa de implementación, se concretó la lógica completa dependiente de los algoritmos trabajados en las secciones anteriores. Se enlazaron los resultados de cada algoritmo como entrada para la siguiente etapa, y se estableció un límite de repeticiones para llevar a cabo el mapeo.

La primera etapa de la lógica completa, está basada siempre en la obtención de un mapa local a través del módulo de sensores. Este mapa local, se toma como entrada junto con el mapa acumulado hasta el momento, y se mandan al algoritmo de cambio de coordenadas, en el cual se traslada el mapa acumulado al nuevo punto de referencia. Una vez trasladado el mapa acumulado, se mandan ambos al algoritmo de alineamiento y coincidencia, en donde se corrigen errores y ruido y se logra localizar de una manera exacta la posición del robot. Con los mapas alineados, se procede a ejecutar el algoritmo de unión y eliminación de repeticiones, en el cual se genera el mapa global actualizado, y es en base a este que se ejecuta el algoritmo de exploración. Este algoritmo de exploración, va a determinar la siguiente mejor posición para incrementar el conocimiento del ambiente basando su lógica en el criterio implementado. Al finalizar la ejecución de este algoritmo, se exportan los datos al módulo de visualización en 3D a

través de un archivo de texto que contiene las coordenadas cartesianas del mapa. Una vez seleccionada la siguiente posición, se le indica al robot que ejecute el movimiento.

Este proceso se repite un número determinado de veces. Este número está de acuerdo a las dimensiones estimadas del ambiente a explorar, y el sector que revela un escaneo del sensor. Esta lógica, se puede sintetizar en el siguiente diagrama de flujo:

FIGURA NO. 191. LÓGICA COMPLETA DEL MÓDULO



Es importante mencionar, que el diagrama de la Figura No. 191 representa la lógica base del módulo. Como se discutirá posteriormente, esta lógica se modificó para adaptarse a distintas situaciones en la implementación real.

2. RESULTADOS

FIGURA NO. 192. LÓGICA DEL MÓDULO EN SIMULACIÓN - 5 MOVIMIENTOS

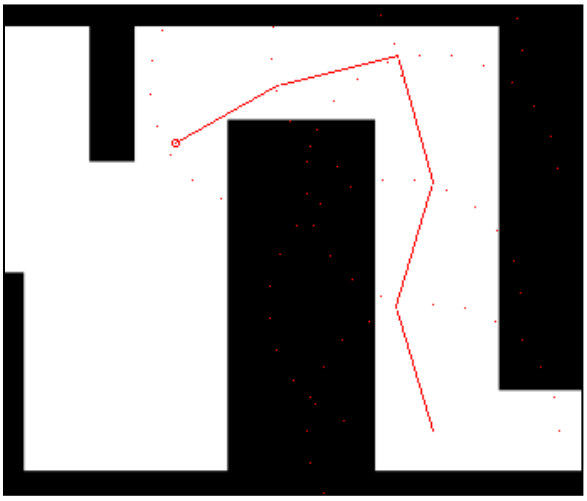


FIGURA NO. 193. MAPA OBTENIDO EN SIMULACIÓN - 5 MOVIMIENTOS

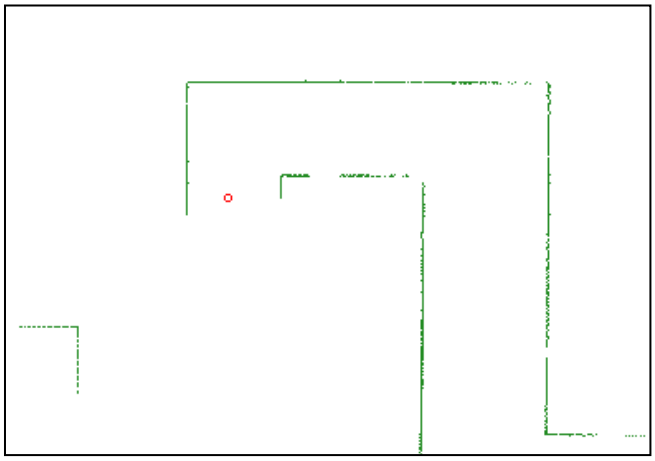


FIGURA NO. 194. MAPA AMBIENTE REAL 1

FIGURA NO. 195. MAPA PASILLO LABORATORIO DE ING. MECATRÓNICA, UVG

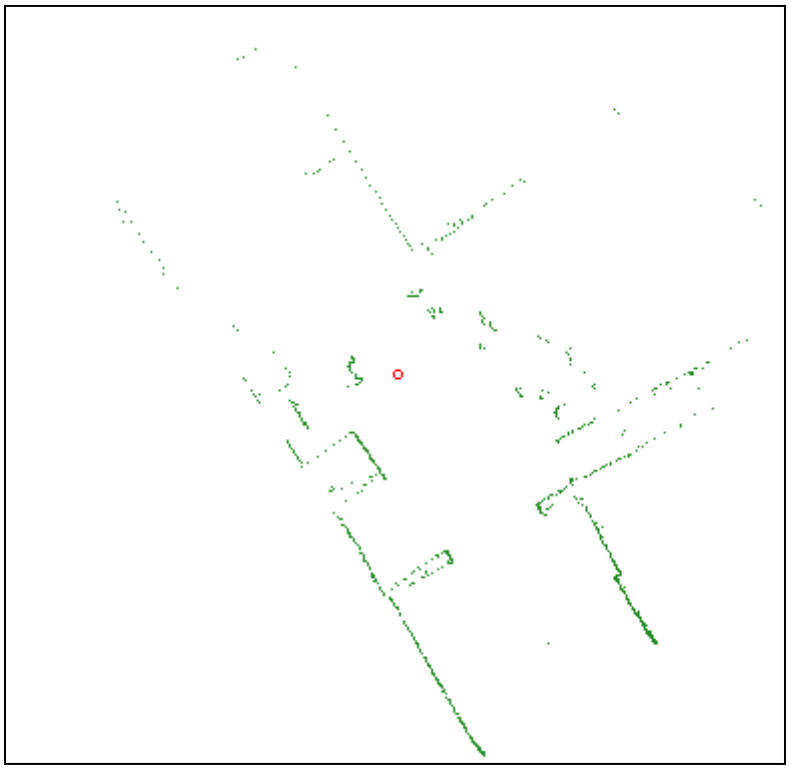


FIGURA NO. 196. PASILLO TERCER PISO EDIFICIO J, UVG

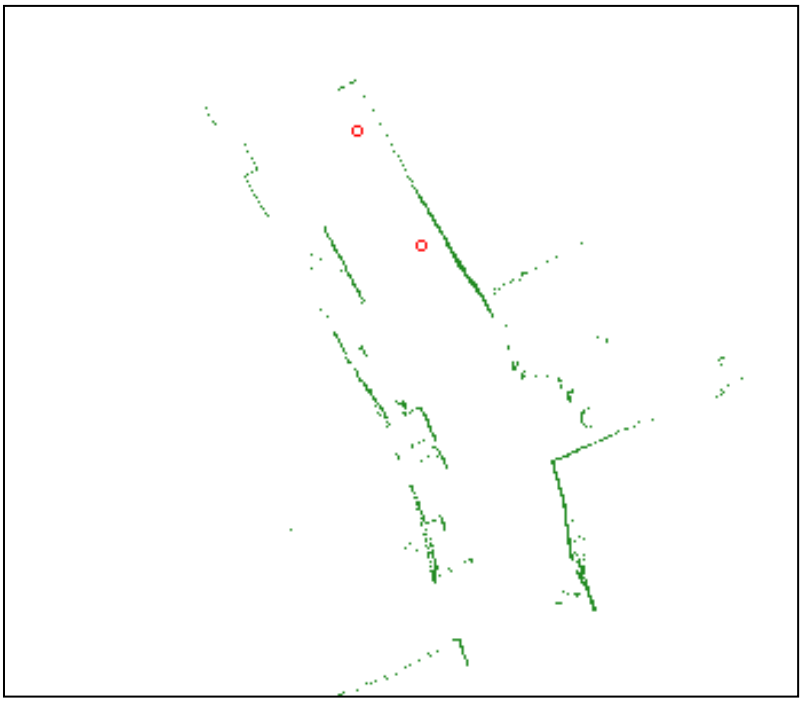


FIGURA NO. 197. PASILLO EXPLORADO

FIGURA NO. 198. PROTOTIPO UTILIZADO PARA GENERACIÓN DE MAPAS REALES

3. DISCUSIÓN. En la Figura No. 192 y Figura No. 193, se muestran los resultados de la aplicación de la lógica completa del módulo en el ambiente de simulación. Se puede apreciar el correcto funcionamiento de los algoritmos de construcción de mapas, como también la lógica de exploración. En pocos movimientos, se recorrió una parte significativa del ambiente de simulación.

Estos resultados en la simulación, dieron la pauta para proceder con la implementación real, y los primeros resultados de la misma se muestran en la Figura No. 194. Este mapa fue generado sin la lógica de exploración. Simplemente se fue moviendo y rotando el sensor dentro del ambiente, y después de 100 movimientos, se obtuvo el mapa mostrado.

Para obtener los mapas presentados en la Figura No. 195 y Figura No. 196, se construyó un robot con el kit LEGO Mindstorm, para poder hacer pruebas con la implementación completa del algoritmo. Este robot experimental implementa un sistema de odometría interno, permitiéndole al algoritmo tener información de la traslación

teórica entre posiciones de observación. El robot de prueba se muestra en la Figura No. 198.

Es importante mencionar que con la aplicación experimental, se notaron ciertas limitaciones del módulo. Una de las principales limitaciones es que los movimientos del robot no pueden ser bruscos. Si el robot rota más de 60º entre escaneos del ambiente, el algoritmo de alineamiento y coincidencia converge a una solución errónea. En movimientos de traslación, es un poco más flexible, pero siempre se obtienen mejores resultados cuando la traslación entre escaneos es pequeña.

Al momento de implementar en el robot del megaproyecto la lógica del módulo, se tuvo que aplicar modificaciones a la lógica del módulo para permitir una integración funcional.

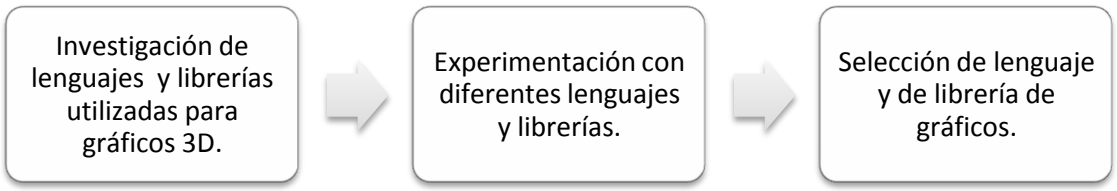
Una de las modificaciones que cabe mencionar en este trabajo, concierne al algoritmo de alineamiento. Este algoritmo, como se discutió previamente, provee un valor de ajuste que corrige el error entre mapa nuevo y mapa acumulado. Este valor de ajuste, en unión con la traslación teórica del robot, da un valor muy exacto de la traslación real del robot. En la integración de todos los módulos del megaproyecto SEEQ, se hizo necesario utilizar el dato obtenido por el algoritmo de alineamiento para mantener informado al robot de su posición. Este dato contribuyó a la estabilización del robot, además de formar parte importante de la lógica de movimiento. Para que el dato de posición fuera útil para el robot, se tuvo que mejorar el tiempo de ejecución del algoritmo. Esto se logró simplificando la interfaz del programa, y optimizando la recepción de datos del sensor. Las modificaciones realizadas se detallan en el capítulo de integración de módulos.

Es importante mencionar que la solución planteada en este proyecto, se limita a la exploración y generación de mapas de ambientes interiores, en donde siempre se detecta algún segmento del ambiente. Si el sensor no detectará el contorno del ambiente, no se tendría ninguna información de la posición del robot, y se perdería la funcionalidad de la generación de mapas, y por ende la exploración no tendría una base de información para tomar decisiones.

XI. VISUALIZACIÓN EN 3D

A. DETERMINACIÓN DE LENGUAJE Y LIBRERÍAS PARA GRÁFICOS 3D

1. DISEÑO EXPERIMENTAL



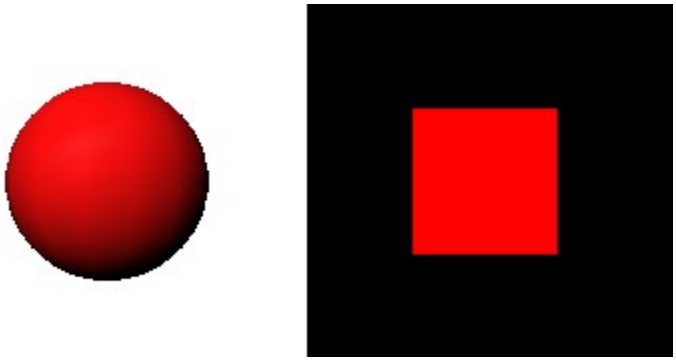
2. RESULTADOS. En la primera fase de la búsqueda de lenguaje, se exploraron todas las opciones que se tenían. Se experimentó C++, Java3D y por último C#.

Originalmente se encontró un motor 3D llamado "3D Developer Studio". Esta opción ofrecía muchas ventajas, como velocidad, simplicidad, etc. Al momento de comenzar a trabajar con ella en C++, se notó que la documentación encontrada, no estaba totalmente actualizada, por lo que muchas cosas, no funcionaban como se decía. Este resultado provocó que se buscara otra opción totalmente diferente.

En la siguiente etapa, se comenzó a trabajar con Java3D. Este es un marco de trabajo, que se fundó sobre las librerías de OpenGl y DirectX. Esta opción, también ofrecía una interfaz simple y libre de descarga.

Al realizar la instalación y las primeras pruebas de dibujó, se notó que se estaba trabajando con algo que era más simple que lo anterior y que además tenía mejor documentación. En la Figura No. 199 se pueden observar las primeras de dibujo e iluminación, que se realizaron.

FIGURA NO. 199. PRIMERAS PRUEBAS UTILIZANDO JAVA3D



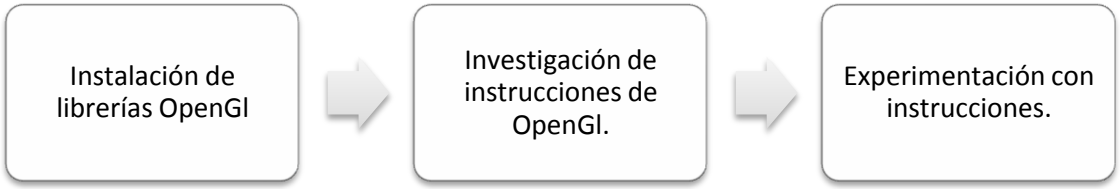
3. DISCUSIÓN. Aunque ya se había empezado a dibujar sobre la plataforma de Java3D, se decidió utilizar el lenguaje C#. Esta decisión se basó en que el módulo de vuelo automatizado y el de sensor láser, trabajaron sobre C#. Por lo que fue necesario hacer el cambio, para lograr una mejor unión de código en un futuro.

El lenguaje C# ofrece las ventajas de que es bastante conocido y completamente gratis. Además para aplicaciones 3D, puede trabajar utilizando OpenGl o DirectX.

Como primera opción, se optó por las librerías OpenGl, ya que son las más antiguas, por lo que se pensó que se iba a encontrar una mejor documentación que utilizando DirectX. Además OpenGl es una librería que no se actualiza tan frecuentemente como DirectX, por lo que probablemente la información que se encontrara no iba a ser obsoleta y se iba a implementar fácilmente.

B. GRÁFICOS EN OPENGL

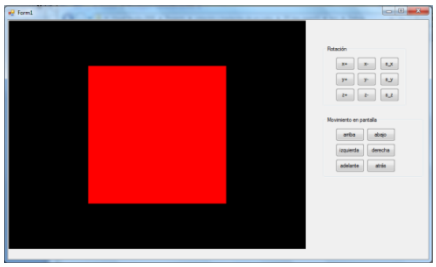
1. DISEÑO EXPERIMENTAL



2. RESULTADOS. Para utilizar las librerías de OpenGl en C# el primer paso fue descargar e instalar TaoFramework (librería que permite utilizar OpenGl en el lenguaje C#).

La primera idea que se tuvo, fue que la información del sensor laser, debía graficarse por medio de cubos pequeños, posicionados en los lugares en donde el sensor encontrara un obstáculo. Esto llevó a que la primera experimentación fuera realizar un cubo. Este cubo se implementó por medio de 6 polígonos utilizando la función GL_QUADS. En la Figura No. 200 se puede observar que aunque se dibujó un cubo, sólo se puede apreciar una cara del mismo, es decir, no se puede observar que es una figura en tres dimensiones.

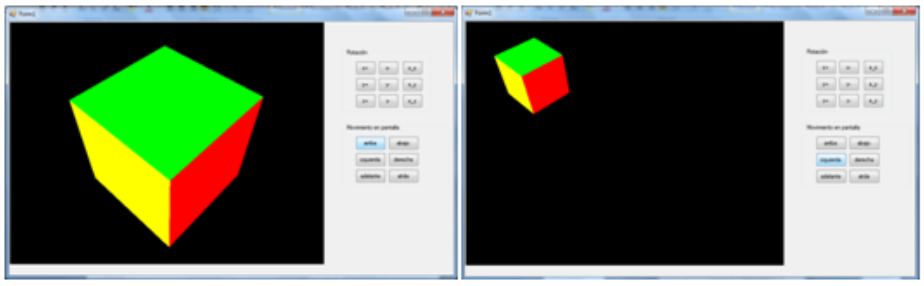
FIGURA NO. 200. CUBO DIBUJADO EN OPENGL



Para notar que lo que se había dibujado era una figura tridimensional, fue necesario rotar la figura. En OpenGl existen funciones que se utilizan para hacer una rotación o traslación. Estas se aplican antes de dibujar, ya que su función es rotar o trasladar el mundo, no una figura específicamente, ni la cámara.

En la Figura No. 201 se puede observar una imagen tridimensional, en la imagen de la izquierda sólo se aplicó una rotación al cubo y en la de la derecha se aplicó una traslación y una rotación.

FIGURA NO. 201. ROTACIÓN Y TRASLACIÓN DE UN CUBO DIBUJADO EN OPENGL

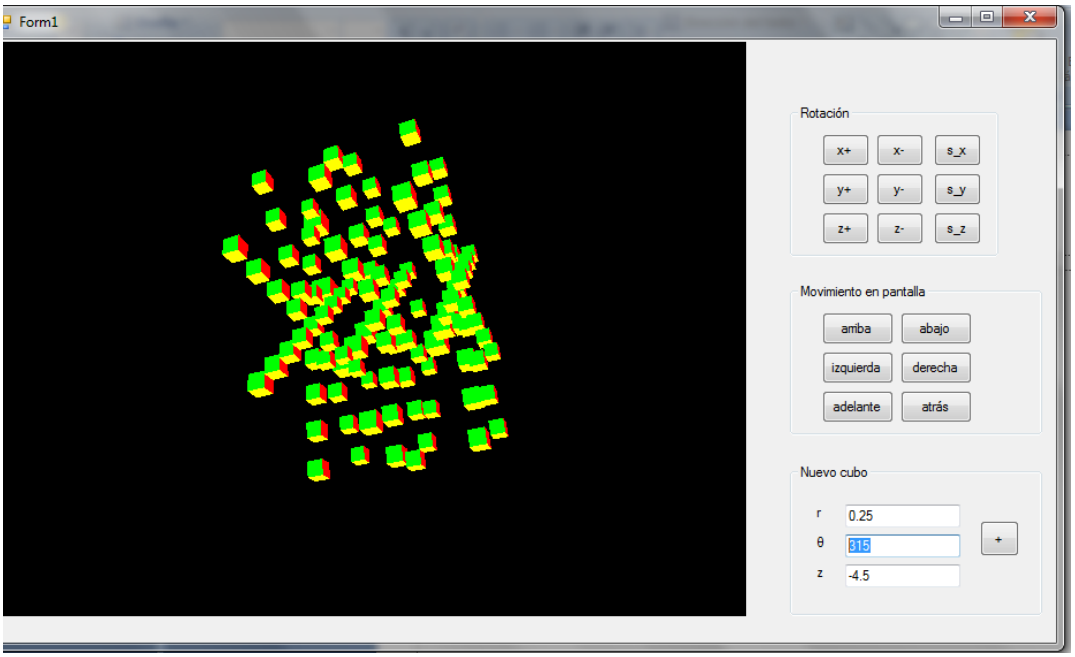


El siguiente paso fue agregar más cubos al mundo creado, al realizar esto se pudo notar que el primer objeto se dibuja con referencia al centro del mundo. El segundo objeto que se dibuja, toma de referencia el centro del que se dibujó anteriormente. Esto pasa de esta manera ya que para dibujar cualquier figura en otra posición era necesario hacer primero una traslación hacia el punto donde se quería dibujar.

Para realizar un método de dibujo más general se utilizaron traslaciones antes y después de dibujar. Antes para llegar al punto de dibujo, y después para regresar al punto de origen. De esta manera nunca se perdería el punto inicial de referencia.

Este método general se trabajó para que dibujara un cubo en la posición que se le indicara, dando la posición en coordenadas polares. En la Figura No. 202 se muestra que utilizando este método agregaron todos los cubos deseados en la posición que se indicara.

FIGURA NO. 202. VARIOS CUBOS DIBUJADOS EN OPENGL



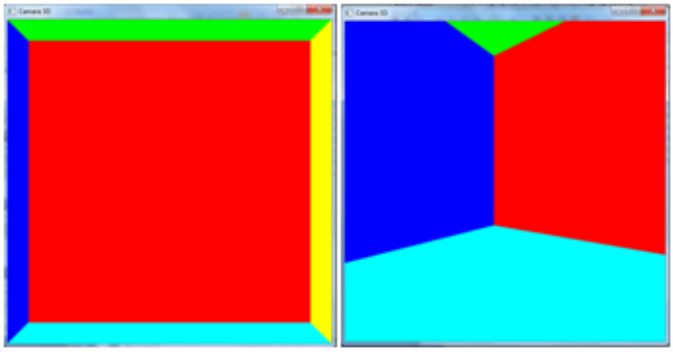
En la fase anterior las rotaciones se encargaban de rotar el eje de coordenadas del mundo, por lo que cada rotación se daba alrededor del cero inicial. En este momento se pensó que sería más natural mover la cámara para navegar en el mundo, en lugar de mover todo el mundo y dejar la cámara estática.

Se investigó acerca de mover la cámara pero las librerías de TaoFramework, no ofrecen esa función, por lo que se recurrió a las librerías de Glut, las cuales son librerías auxiliares de TaoFramework, que ofrecen funciones extra.

En la siguiente fase se realizó los controles de traslación y rotación utilizando las librerías de Glut para mover la cámara y la dirección de vista de la misma. Para realizar las pruebas, se dibujó un solo cubo que abarcara toda la vista de la cámara, de manera de simular un cuarto para observar los movimientos dentro del mismo.

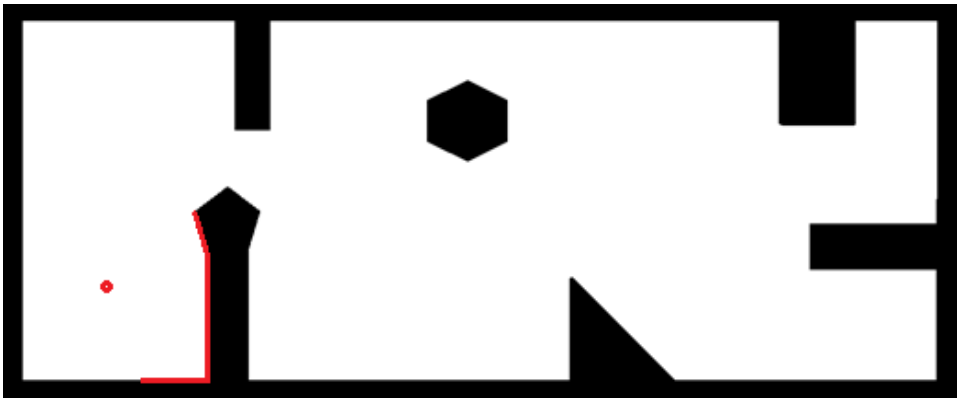
En la Figura No. 203 se puede observar una vista del cuarto que se simuló y la rotación dentro del mismo.

FIGURA NO. 203. VISTA DE ROTACIÓN Y TRASLACIÓN DENTRO DE UN CUBO



La siguiente fase fue graficar datos obtenidos de una simulación de mapeo del módulo de vuelo automatizado. En la Figura No. 204 se puede observar la simulación utilizada. El punto rojo representa el sensor y las líneas rojas representan los datos escaneados por el sensor.

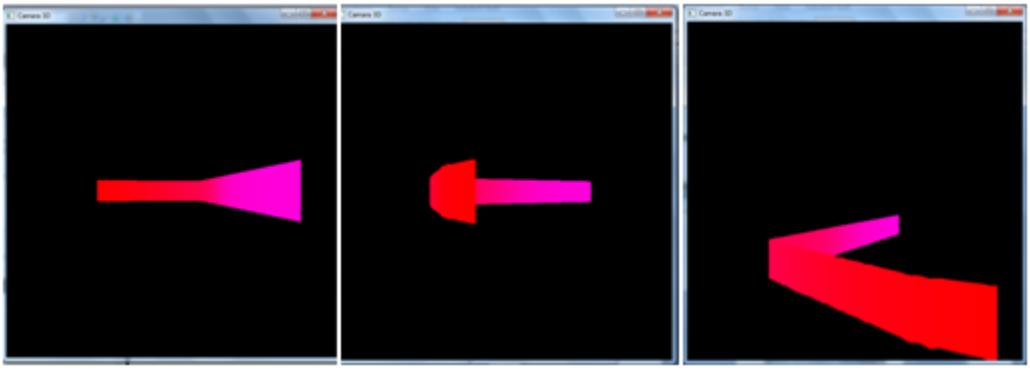
FIGURA NO. 204. SIMULACIÓN UTILIZADA PARA PRIMER DIBUJO DE MAPA



Los datos de la simulación se obtuvieron en un archivo de texto, cada línea contenía un par ordenado en coordenadas polares. Toda línea que contuviera un valor de -1en el radio, representaba que en ese ángulo no se tenía dato, o se tenía un valor inválido del mismo.

Para dibujar se fueron tomando los datos cada dos pares de coordenadas, si una de estas contenía un radio de -1 no se graficaba el polígono. En la Figura No. 205 se puede observar el resultado de la gráfica de los datos simulados. Si se compara con la Figura No. 204, la cual es la simulación, se puede observar que se obtuvo un gráfica completamente representativa de los datos simulados.

FIGURA NO. 205. RESULTADO DE PRIMER MAPEO



Para cada polígono que se dibujó se fue cambiando el color utilizado, por lo que es posible observar que la cantidad de datos utilizados proporciona una resolución suficiente para construir un mapa representativo.

En esta etapa se utilizó una altura predefinida, ya que de la simulación sólo se obtuvieron datos en dos dimensiones.

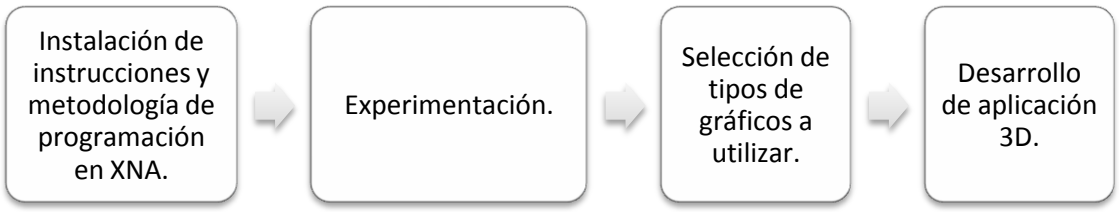
3. DISCUSIÓN. En esta etapa del proyecto se notó que el manejo de la cámara no era correcto. Al hacer más pruebas y navegar dentro de diferentes mapas muchas veces era difícil encontrar todas las partes del mismo.

Debido a falta de documentación de este tipo de problemas, se decidió probar con otro lenguaje, DirectX. Aunque se podría decir que ambos son básicamente lo mismo, DirectX ofrece un conjunto de herramientas llamado XNA, el cual se puede utilizar en el

entorno seleccionado (C#). XNA facilita muchos procesos, como el manejo de la cámara, los cuales pueden llegar a ser bastante complicados si no se tiene el conocimiento suficiente. XNA facilita estos procesos, debido a que trabaja en un nivel más alto de programación que DirectX y OpenGl (los cuales se encuentran en un mismo nivel), por lo que ofrece una curva más corta de aprendizaje, y facilita la experimentación y el manejo del mismo.

C. GRÁFICOS EN DIRECTX CON XNA

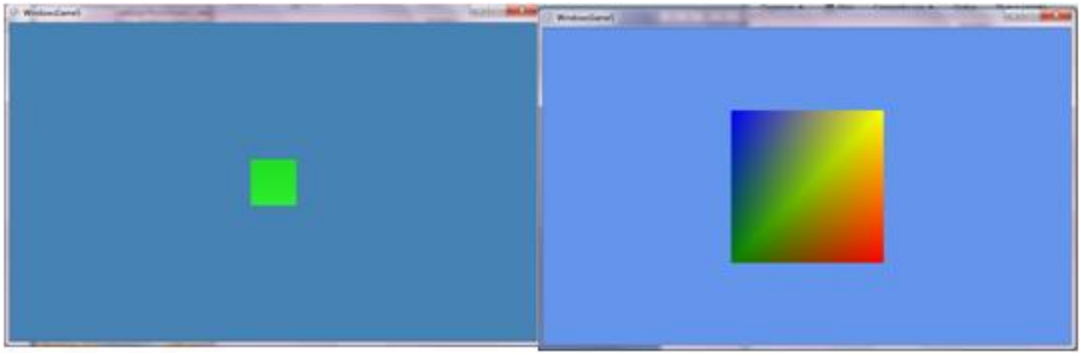
1. DISEÑO EXPERIMENTAL



2. RESULTADOS. La teoría básica de gráficos en tres dimensiones aprendida al utilizar OpenGl, pudo ser aplicada al graficar con XNA. Sin embargo las instrucciones y estructura general del programa son completamente diferentes, por lo que se tuvo que realizar la experimentación desde el principio.

Se comenzó en aprender a inicializar el mundo, posicionar la cámara y a dibujar una figura primitiva. Al igual que en OpengGl, las figuras primitivas que se dibujaron, tienen información de posición y color para cada vértice. En la Figura No. 206 se puede observar el primer cuadrado dibujado en XNA.

FIGURA NO. 206. PRIMEROS CUADRADOS DIBUJADOS EN XNA



Para ambas imágenes de la Figura No. 206 fue necesario dibujar dos figuras primitivas, ya que el triángulo es el elemento esencial para toda gráfica en XNA.

El siguiente paso fue empezar a manejar la cámara de la manera más adecuada, para que al moverse dentro del mundo, el movimiento fuera lo más cercano a la realidad posible. En esta etapa se hicieron varias pruebas, para cerciorarse que el movimiento de la cámara era el adecuado. Al final de esta etapa la cámara se controló por medio de una forma llamada "cámara de primera persona". Este es el tipo de cámara que se utiliza en la mayoría de juegos.

Cuando se consiguió un control adecuado de la cámara, se realizó la primera prueba de mapeo con datos obtenidos del módulo del vuelo automatizado. Esta vez los datos correspondieron a una prueba real utilizando el sensor, y no una simulación, como en la etapa anterior.

Cuando se hizo esto en OpenGl no se consideró que se tendría más de una escaneada y que los puntos no estarían en orden en cómo se escanearon, por lo que en lugar de unir cada plano dibujado con el plano anterior, se dibujó un plano en cada punto de un ancho determinado y de una altura determinada. El mapa que resultó de esta prueba fue bastante representativo del lugar en el que se estaba escaneando. Se notó que las partes más continuas del mapa correspondían a los puntos en donde el sensor había escaneado más veces, o más cerca, y por lo tanto regresaba datos más cercanos entre sí.

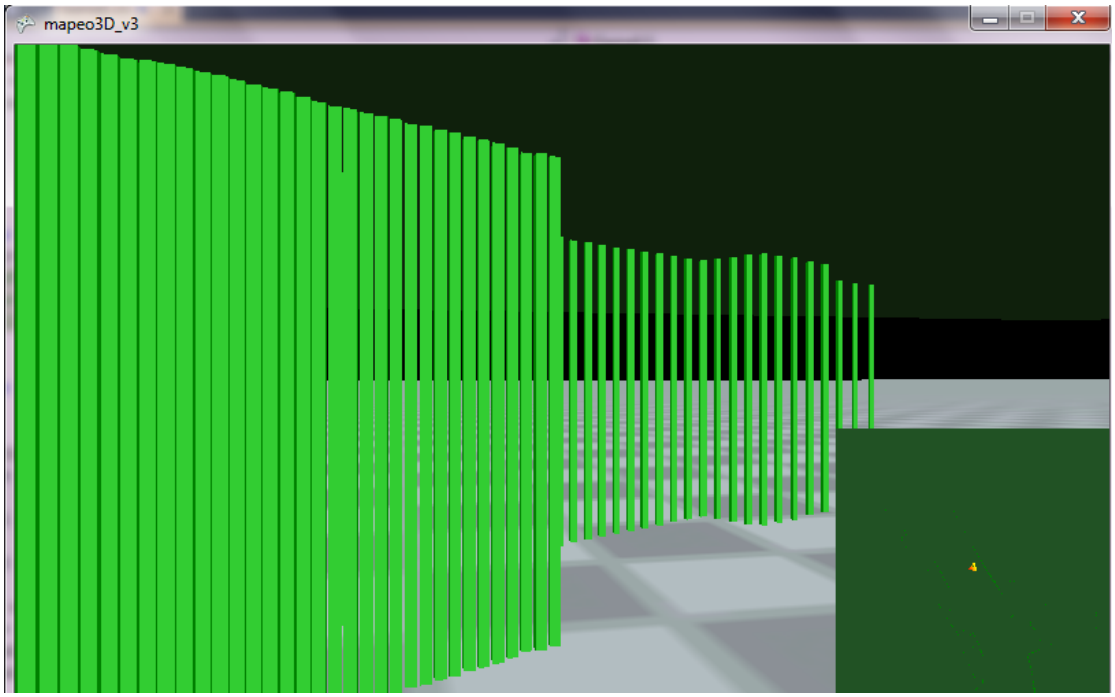
Después de finalizar esta vista del mapa, se pensó que sería de bastante ayuda para el usuario, tener una visualización de planta del mismo, y a la vez, observar el punto en donde se encontraba dentro del mapa.

Para esta visualización se utilizó una cámara de tipo ortográfica, para que no existiera ninguna deformación de lo que se dibujó. En esta etapa fue necesario cambiar los planos dibujados en cada punto por columnas. Esto se hizo para que el mapa se viera desde la vista ortográfica de planta.

Realizar una columna por punto a graficar, generó un código más extenso, y además cargo bastante más el buffer de datos, que cuando se dibujaban sólo planos. La razón de esto fue que para dibujar un plano se necesitan de 6 vértices y para dibujar una columna se necesita al menos cinco planos, sin contar el inferior, por lo que cada columna necesita un total de 30 vértices.

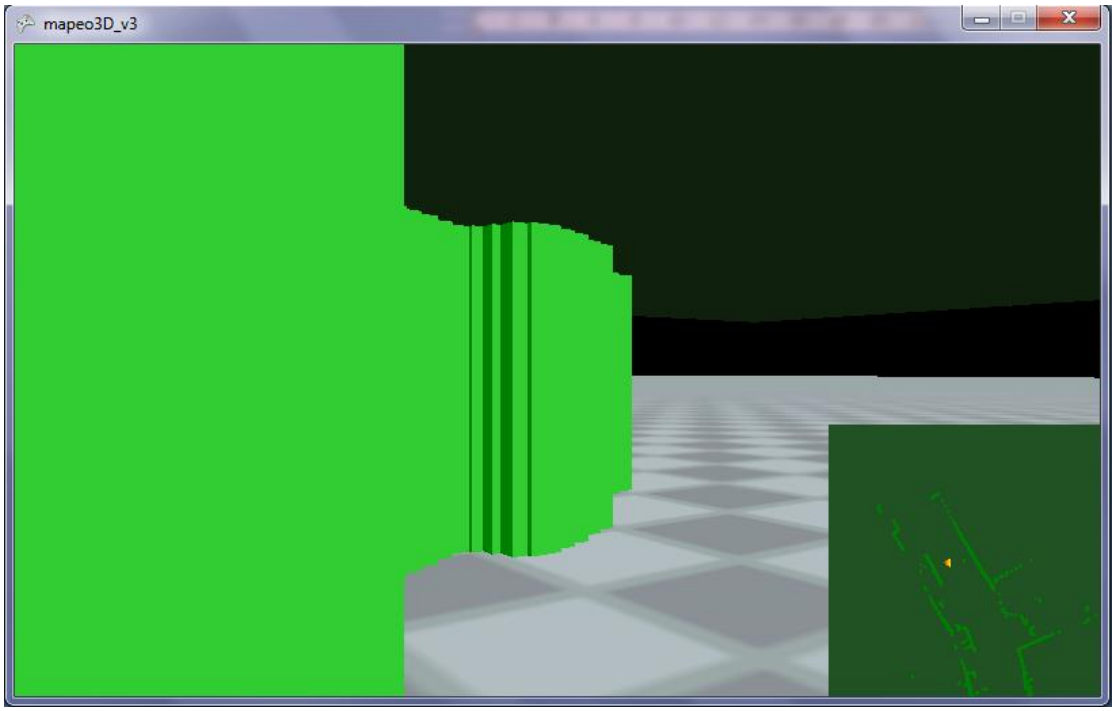
En la Figura No. 207 se puede observar el resultado de esta prueba, se puede observar que se generó una vista aceptable del mapa ortográfico, pero hizo que los datos se cargaran de una forma bastante lenta.

FIGURA NO. 207. PRIMER MAPEO EN XNA UTILIZANDO FIGURAS PRIMITIVAS



La siguiente mejora que se hizo, fue aumentar el ancho a cada columna, ya que habían muchas separaciones entre cada una de ellas y el mapa de planta no se puede apreciar por completo. En la Figura No. 207 las columnas tienen un ancho de 10cm, por lo que para la siguiente prueba se utilizó un ancho de 100 cm. Este ancho se ajustó utilizando varios mapas, hasta alcanzar el óptimo para la aplicación.

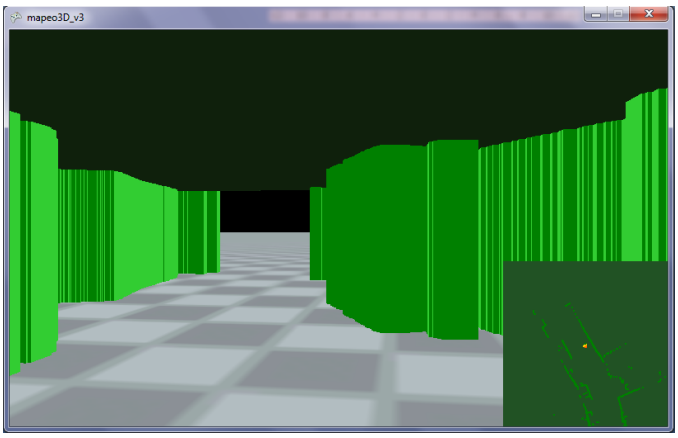
FIGURA NO. 208. MAPEO CON PRIMITIVAS DE ANCHO MAYOR



En la Figura No. 208 se puede observar que el resultado que se obtuvo de la pared fue continuo. En el mapa de planta se puede apreciar mayor detalle y claridad que en la prueba anterior.

En medio del mapa proyectado se puede observar un pequeño triangulo amarillo y rojo, este sirve para indicar la posición de la cámara dentro del mapa. El vértice de color rojo apunta en la dirección en la que observa la cámara. En la Figura No. 209, se puede observar otra vista del mapa de la Figura No. 208.

FIGURA NO. 209. SEGUNDA VISTA DEL MAPEO CON PRIMITVAS DE ANCHO MAYOR

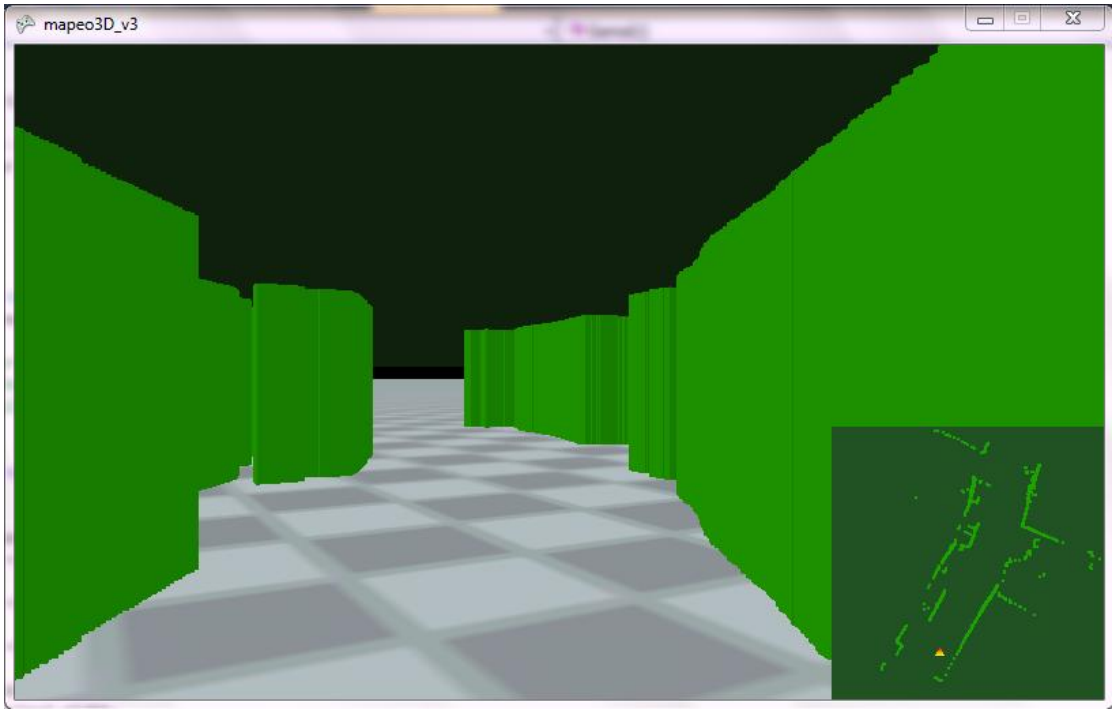


No se incrementó el ancho de cada columna, ya que el mapa se empieza a deformar, y se unen partes que no pertenecen juntas. Además esto reduce el ancho de los caminos del mapa, ensancha las paredes y se obtiene un resultado distorsionado de la realidad.

El siguiente paso fue conectarse con el programa que recibe los datos escaneados por el sensor laser y hacer un mapeo en tiempo real. Se pudo notar que este tipo de gráficos son pesados para el dispositivo de gráficos de la computadora que se utilizó, ya que cada vez que se realizaba una actualización del mapa, es decir, cada vez que se obtenían nuevos datos del mismo, el proceso de, borrar el mapa actual, y graficar los nuevos datos retrasaba el tiempo de dibujo. Este tiempo depende significativamente de la cantidad de datos a graficar, ya que por cada punto se debían de realizar 10 figuras primitivas, lo que significa guardar 30 vértices en el buffer de vértices, que es lo que se utiliza a la hora de graficar. La mayoría de los mapas que se probaron tenían al menos 1000 puntos, por lo que el buffer de datos tendría que tener un tamaño de 30,000 datos.

Para mejorar la respuesta en el tiempo se dibujó con modelos 3D. Los modelos se realizaron en un software de AutoDesk llamado Maya. Este software se especializa en realizar modelos 3D, los cuales se pueden convertir al formato que utiliza XNA, el cual es “fbx”. Se realizó un modelo con forma de columna de una altura de 1 metro y de un ancho de 100 por 100 cm. Para graficar el mapa se colocó una columna por cada dato que dio el sensor. En la Figura No. 210 se puede observar el resultado final obtenido utilizando modelos 3D.

FIGURA NO. 210. MAPEO UTILIZANDO MODELOS 3D



Se puede ver que es casi lo mismo que se había logrado con figuras primitivas, pero el procesamiento involucrado para graficar cada punto es mucho menor, por lo que se observó una mejora significativa en tiempo de dibujo en pantalla. Al graficar los datos del mapa en tiempo real utilizando modelos 3D, no se percibió ningún paro en el movimiento de la cámara, como se podía apreciar con las figuras primitivas para dibujar los nuevos datos obtenidos del archivo de texto.

Este tipo de dibujo en definitiva es más rápido de enviar por el dispositivo de gráficos y de dibujar en pantalla que la metodología utilizada anteriormente. En la Figura No. 197, se puede apreciar una fotografía del pasillo real que se escaneó con el sensor.

3. DISCUSIÓN. El cambio de OpenGl a DirectX, ayudó a que se comprendiera de una manera más rápida el proceso de graficar en tres dimensiones. Esto se debe a que se encontró mejor documentación de DirectX especializada para el lenguaje en el que se estaba trabajando (C#), que de OpenGl. Las librerías de TaoFramework de

OpenGl para C#, aunque son muy buenas, ya no se utilizan mucho debido a que se detuvo el desarrollo de las librerías originales.

Otro cambio que fue muy importante para obtener una buena aplicación 3D, fue el cambio de utilizar figuras primitivas a utilizar modelos 3D. Esto ayudó ya que los modelos tienen un procesamiento especializado de dibujo el cual se procesa mejor por el dispositivo de gráficos de la computadora, que el proceso de enviar todos los vértices por separado.

Al final se llegó a un mapa que es bastante representativo del lugar que se desea graficar. La decisión de graficar todos los puntos como columnas y no tratar de unirlos con el punto siguiente que se obtiene también fue bastante importante. Esto se debe a que los datos del mapa se obtienen en un orden diferente a los que se obtienen del sensor, por los que los puntos que son consecutivos en el sensor, no necesariamente son consecutivos en los datos que se tienen en el mapa. Por lo tanto los puntos consecutivos del mapa, no necesariamente se deben graficar unidos.

D. COMUNICACIÓN INALÁMBRICA

1. DISEÑO EXPERIMENTAL. En esta parte del proyecto se pensó que las aplicaciones trabajadas en la computadora (la parte de vuelo automatizado y la de mapeo 3D), iban a estar fusionadas en una sola aplicación. Y se pretendía que existiera una comunicación de doble vía con el helicóptero, que se pudieran recibir los datos del sensor, y que se le dijera a qué posición moverse desde la aplicación 3D. Esta comunicación se iba a realizar por medio de los módulos Xbee.

2. RESULTADOS. En el transcurso del proyecto, se notó que el helicóptero necesitaba ayuda a estabilizarse. Por lo que el módulo de vuelo automatizado tuvo que cambiar para enviarle datos de posición. Este proceso consumió demasiados recursos de la computadora en donde se encuentra el módulo de vuelo automatizado, por lo que se decidió trabajar el módulo de la aplicación 3D por aparte.

Sin embargo al dejar la aplicación 3D en otra computadora, surgió la necesidad de que existiera una comunicación con la computadora del módulo de vuelo automatizado. Esta comunicación fue necesaria, ya que el módulo de vuelo automatizado, es el primero que obtiene los datos del sensor, para crear un solo mapa y poder decirle al helicóptero cual debería de ser su siguiente mejor posición.

Esta comunicación se realizó por medio de una carpeta compartida que se actualiza por internet. Dentro de esta carpeta se puede encontrar un archivo de texto, el cual contiene la información del mapa. La información del mapa que se obtiene, contiene un punto en coordenadas cartesianas por línea. Al recibir el archivo de texto, el programa de gráficos, lee línea por línea y posiciona una columna en cada punto encontrado en el archivo de texto. Aunque este método parece bastante ineficiente, se debe recordar que la lectura y escritura de archivos de texto, es un procedimiento bastante ligero, es decir no consume muchos recursos, en términos de memoria ni tiempo. Por lo que aunque parezca algo tedioso y largo, archivos con una cantidad mayor a 1,000 puntos se procesan en cuestión de milisegundos.

3. DISCUSIÓN. La estabilización final del helicóptero se logró por medio de un control de posición que se realiza en el módulo de vuelo estabilizado. Este control y todo el procesamiento de este módulo, no permitió que se crearan los gráficos en la misma aplicación, ya que estos también consumen muchos recursos de la computadora.

Por esta razón fue necesario implementar otro tipo de comunicación entre la computadora del módulo de vuelo automatizado y la computadora del mapeo en tres dimensiones. Se decidió utilizar una carpeta compartida actualizada por medio de internet ya que de esta manera el programa del módulo de vuelo automatizado, se interrumpe lo menos posible.

Esta forma resultó bastante conveniente para ambas aplicaciones, ya que esta comunicación se realiza todo el tiempo, y el programa no invierte tiempo en enviar ni recibir datos de otro lado, en cambio sólo fue necesario que regularmente se leyera el archivo de texto, para actualizar los datos del mapa.

Otra ventaja muy importante de este tipo de comunicación, es que la distancia entre el helicóptero y la computadora que tiene este módulo no es importante. Sólo es importante que la computadora del vuelo automatizado y la del mapeo tengan acceso a internet. Este resultado significa que la aplicación 3D puede verse en cualquier parte del mundo mientras se está actualizando el mapa, y además que puede verse en varias computadoras al mismo tiempo.

Aunque se puede observar que este método trajo bastantes ventajas, también se observaron algunas desventajas que podrían ser corregidas. La primera encontrada es que se depende completamente de internet y del programa que se utilizó para hacer las carpetas compartidas; en este caso Dropbox. Es decir, si en algún momento se pierde la conexión a internet, o se tienen fallas en el sistema de Dropbox, la aplicación 3D fallaría en mapear los datos en tiempo real. Otra desventaja es que se pierde mucho tiempo en el proceso, por lo que el mapeo no se encuentra lo más actualizado que podría estar. Se pierde tiempo, ya que primero, el módulo de vuelo automatizado reescribe el archivo de texto, después este se sube a internet, pasa por el sistema de dropbox, el cual lo distribuye a todas las personas que compartan la carpeta del mapeo, y finalmente se descarga en la computadora en donde se tiene la aplicación 3D, dejando el archivo disponible para la lectura del mismo.

Este proceso podría cambiar bastante si se hubieran utilizado otros recursos como las bases de datos. Con una base de datos que se tenga en un servidor local o por medio de internet, se podrían haber realizado procedimientos más simples y más eficientes que con la actualización por medio de carpetas compartidas. Por ejemplo, no hubiera sido necesario borrar todos los datos anteriores para agregar los nuevos puntos que se hayan escaneado, ya que a una tabla de una base de datos se pueden agregar elementos hasta el final de la misma. Aunque esto se podría haber realizado de igual manera en un archivo de texto, las bases de datos se especializan en el manejo de los mismos, por lo que tienen métodos mucho más eficientes de guardar y recopilar los datos de interés.

Las bases de datos también ayudarían a que se independice más la aplicación y no tenga que pasar por tantas etapas antes de llegar al destinatario.

Estos beneficios podrían haber sido de gran ayuda para tener un mapa más actualizado en todo momento, pero debido a los alcances especificados en este proyecto, esto sólo se queda como una recomendación para un trabajo que quiera continuar con un proyecto similar al presente.

XII. INTEGRACIÓN DE MÓDULOS

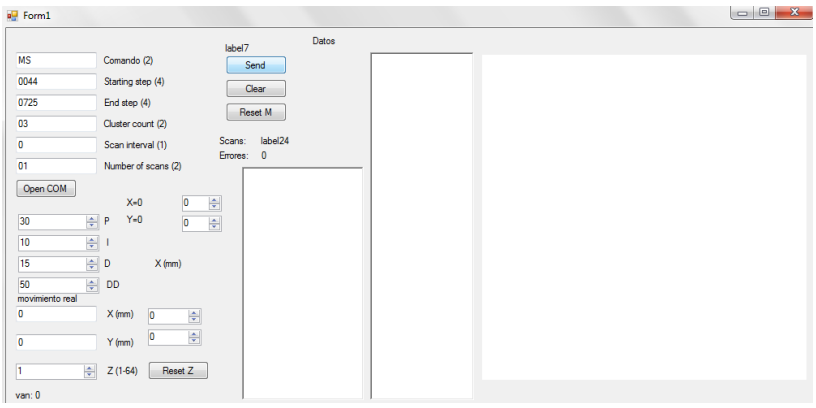
A. CONTROL AUTÓNOMO DE CUADCÓPTERO

1. DISEÑO EXPERIMENTAL. Para esta etapa final del proyecto, se unificaron todos los módulos desarrollados en los capítulos anteriores, y se llegó a la implementación definitiva. Se realizó el control de posición del cuadcóptero utilizando el sensor laser, el algoritmo de localización y un control PID de segundo orden. Como se ha mencionado con anterioridad, el sensor láser permite obtener un escaneo del entorno donde se encuentra el cuadcóptero y por medio del algoritmo de localización se crea un mapa interno que permite, no solamente conocer los objetos cercanos, sino que también la posición y orientación del vehículo aéreo.

Una vez obtenida la medición de posición se procedió a realizar el control de desplazamiento lineal. Para dicho control se utilizó el software de C#; se realizó un hilo (*thread*) dentro del cual se obtiene la lectura de posición actual, se obtiene el error respecto a la posición deseada y se implementa un control PID de segundo orden cuya respuesta es enviada por medio de un módulo de radio frecuencia hacia el cuadcóptero. Dicha señal enviada como resultado del PID afecta al control local de inclinación en el cuadcóptero, produciendo así mayor o menor fuerzas horizontales que frenen o aceleren el desplazamiento horizontal.

2. RESULTADOS

FIGURA NO. 211. INTERFAZ FINAL DE APLICACIÓN DE CONTROL AUTÓNOMO



3. DISCUSIÓN. Uno de los principales retos que presenta el control de posición del cuadcóptero radica en la obtención de una medición de posición lo suficientemente precisa y rápida como para poder controlar un vehículo aéreo. Cuando el cuadcóptero se desplaza en el aire sin control, una pequeña inclinación permite que el cuadcóptero acelere y que su velocidad incremente, al realizar un control de posición se debe poder predecir dichas velocidades y aceleraciones lo suficientemente rápido como para que se compense el movimiento antes que sea demasiado tarde y el vehículo se encuentre en un lugar no deseado. En la implementación del control de posición presentada en este proyecto se requiere un algoritmo para detectar el estado del vehículo dentro de un ambiente. Este algoritmo toma cierto tiempo ya que debe ser lo suficientemente robusto para no perder la orientación y posición aun cuando el cuadcóptero se encuentre en movimiento. De esta cuenta se obtiene un dato de posición nuevo 4 veces por segundo, que para la aplicación es lento. El control PID de segundo orden logra mejorar la respuesta, sin embargo, no lo suficiente como para llevar al cuadcóptero a una posición exacta, por lo cual la posición horizontal presenta oscilaciones alrededor del punto deseado. Si se incrementan las constantes del PID de manera que la reacción sea más rápida puede ocurrir una inestabilización del sistema, por esta razón es preferible no tener constantes muy grandes aun cuando el resultado no es el óptimo.

La aplicación elaborada, sigue enviando datos a la visualización en 3D, lo cual permite tener una representación actualizada cada 15 segundos del ambiente en exploración.

XIII. CONCLUSIONES

1. El perfil recomendado para un tubo circular de 16 cm de longitud que sostiene un motor en su extremo que gira a 5700rpm, es de 3/4 de pulgada y 1/32 de espesor para evitar vibración excesiva.

2. Para un modelo aerodinámico de una cruz con cuatro rotores en sus extremos, el análisis de esfuerzos y deflexiones resultan despreciables en comparación con resultados presentados por un análisis de vibración. Siendo así los factores de seguridad obtenidos para el análisis de esfuerzos en los perfiles de aluminio y fibra de vidrio es de 58.27 y 1799.85 respectivamente.

3. Es posible utilizar una estructura con aluminio y fibra de vidrio, para elevar un vehículo aerodinámico si no se desea un diseño óptimo y no se cuenta con muchos recursos como para construirlo con fibra de carbono.

4. La fibra de vidrio es un material que se puede trabajar con poco equipo y en poco tiempo, pero siempre hay que tener las debidas precauciones. Posee características importantes para un vehículo aéreo, como lo son su masa y su resistencia.

5. El análisis del sistema del cuadcóptero, aunque no es lineal en ciertas etapas, se puede simplificar si se considera una función de transferencia entre la señal PWM ingresada a los controladores y la fuerza que producen los motores, ya que los controladores hacen lineal la función de transferencia.

6. El control de rotación es independiente al control de altura y al control de inclinación, si la rotación se controla sumando el error a los dos motores que controlan la inclinación de un eje, los cuales deben girar en sentido horario, y restando el error a los dos motores que controlan la inclinación del otro eje, los cuales deben girar en contra de las agujas del reloj.

7. El control de altura es independiente al de inclinación, si este último es realizado sumando el error a un motor y restando el error al otro motor.

8. La frecuencia de corte para el filtro de la señal proveniente de los sensores ultrasónicos, se establece en 3Hz para permitir que la señal presente frecuencias bajas y en especial la señal promedio DC que indica la distancia de un objeto. De esta manera se evitan ruidos ya sea eléctricos, por variaciones de voltaje o mecánicos por cambios abruptos en la medición.

9. Las instrucciones creadas facilitan el control del helicóptero externamente por otros módulos.

10. Un cuadcóptero posee varios ejes de libertad, pero los más importantes para su estabilización son los de movimiento angular, según Brescianni.

11. La información de la inclinación del cuadcóptero es brindada por un acelerómetro de 3 ejes y un giroscopio de 3 ejes. Estos nos brindan información sobre su inclinación por medio de trigonometría y por integración discreta.

12. La utilización de un filtro complementario entre los ángulos obtenidos del acelerómetro y del giroscopio proporciona una mejor aproximación del ángulo, eliminando el sesgo y ruidos, con un tiempo de respuesta bajo.

13. Se logró implementar la interfaz de datos inalámbrica para el radar láser Hokuyo URG-04LX-UG01 de una forma sencilla y robusta.

14. Se lograron implementar todas las instrucciones y comandos definidos en el protocolo de comunicación SCIP en la interfaz de datos inalámbrica.

15. El ruido, errores de medición e incertidumbres son variables que deben ser consideradas para elaborar una estrategia de exploración y generación de mapas funcional.

16. El algoritmo de generación de mapas simulado tuvo problemas de tiempo de ejecución. La respuesta lenta se relaciona con los componentes gráficos de la simulación.

17. El elemento de más importancia en el algoritmo de generación de mapas, es el algoritmo de alineamiento y coincidencia. Este se implementó por medio de una regla de correspondencia entre puntos con restricciones adicionales que hacen óptimos sus resultados.

18. El número de iteraciones del algoritmo de alineamiento y coincidencia se estableció en 20. Este número garantiza buenos resultados en un corto tiempo.

19. Se implementó una fase de omisión de repeticiones para que al incrementar el mapa con nueva información, se mantenga baja la densidad de puntos. Esto permite que el tiempo de ejecución de los algoritmos se mantenga en un rango aceptable.

20. El algoritmo de generación de mapas puede dar información de la localización relativa del robot en el mapa.

21. Se implementó un algoritmo de exploración basado en la información de los mapas generados. El algoritmo permite maximizar el conocimiento de un ambiente de una manera eficiente.

22. Se logró proveer una solución funcional al problema de movimiento autónomo. La solución planteada se limita a la exploración y generación de mapas de ambientes interiores.

23. Las librerías para realizar gráficos en tres dimensiones de XNA, proveen una curva de aprendizaje más corta que las de OpenGl, debido a que trabajan en un nivel más alto de abstracción; esto facilita los procedimientos necesarios para hablar con el dispositivo de gráficos de un computador.

24. Las figura primitivas, consumen más recursos del dispositivo de gráficos que los modelos 3D.

25. Al realizar un mapeo de los datos de un sensor laser, se obtiene una gran cantidad de información de datos puntuales, por lo que para realizar un mapa en tres dimensiones con esta información, es necesario otorgarle volumen a cada punto.

26. El volumen de cada punto que se grafica en el mapa de tres dimensiones debe de ser cuidadosamente calibrado, para no realizar una deformación en el mismo.

XIV. RECOMENDACIONES

1. Es recomendable utilizar perfiles de fibra de carbono si se requiere un diseño optimizado y se tiene suficiente recurso como para invertir en ellos. La fibra de carbono presenta una buena relación entre peso y resistencia.

2. Es recomendable utilizar un controlador y motor que presenten una respuesta transitoria más rápido, para tener una mejor respuesta del sistema de control.

3. Es recomendable utilizar sensores láser en lugar de ultrasónicos para aplicaciones con vehículos aéreos, por su variación en mediciones dinámicas.

4. No se recomienda utilizar un filtro Kalman debido a su complejidad de entendimiento. Sin embargo la obtención utilizando el filtro ponderado fue satisfactoria para esta aplicación y resulta ser una solución más intuitiva.

5. Es aconsejable calibrar los ejes de libertad independientemente. Se observa que el comportamiento general es similar al de estas pruebas, si son realizadas correctamente. Es necesario realizar estas pruebas con la seguridad adecuada.

6. Se recomienda utilizar el compilador C30 de MPLAB para utilizar los microcontroladores de la familia dsPIC30F por ser más robusto y optimizado que otros compiladores en C. También se menciona en la hoja de datos que se encuentra optimizado para el compilador en C.

7. Se debe utilizar protección adicional para realizar las pruebas con el cuadcóptero. Es aconsejable realizarlo acompañado por cualquier inconveniente.

8. Se recomienda utilizar módulos de comunicación inalámbrica con mayor alcance para aumentar la versatilidad de la interfaz de datos del radar láser.

9. Se recomienda mejorar la eficiencia del programa principal del microcontrolador utilizado para lograr transferir los datos a una tasa de transferencia más alta.

10. Se recomienda realizar la decodificación y el procesamiento de los datos en el microcontrolador para aumentar la eficiencia de la interfaz de datos implementada.

11. Es recomendable implementar una aplicación de simulación de ambientes y características reales para sobre ésta desarrollar algoritmos de generación de mapas y de exploración robustos.

12. Se recomienda implementar en el vehículo de exploración técnicas adicionales de odometría, con el fin de ampliar las opciones de aplicación del trabajo desarrollado.

13. Si se busca continuar con esta línea de investigación y desarrollo, se recomienda proceder con la generación de mapas tridimensionales, no a un solo nivel del ambiente. Esto con el fin de crear una representación más fiel y confiable del ambiente, y permitir al módulo de visualización dar más detalle.

XV. BIBLIOGRAFÍA

[1] Achtelik, Markus; Bachrach, Abraham; He, Ruijie; Prentice, Samuel; Roy, Nicholas. 2009. *Autonomous Navigation and Exploration of a Quadrotor Helicopter in GPS- denied Indoor Environments.* Estados Unidos, Massachusetts Institute of Technology.12 páginas.

[2] Amigoni, Francesco; Caglioti, Vincenzo. 2009. *An Information-Based Exploration Strategy for Environment Mapping with Mobile Robots*. Italia, Politécnico de Milano.

[3] Analog Devices. 2010. *Accelometer*. [En línea]. www.analog.com/static/imported-files/data_sheets/ADXL335.pdf

[4] Baker Martin, John. 2010. *Programming Languages for 3d simulation and games*. [En línea]. http://www.euclideanspace.com/software/language/index.htm.

[5] Balachdran, Balakumar. 2006. *Vibraciones*. Editorial Thomson. México, D.F. 581ppt.

[6] Beer, Ferdinan. 2006. *Mecánica vectorial para ingenieros*. 8va Edición. Editorial Mc Graw Hill.

[7] Black-byte. 2006. *Primeros pasos en OpenGl*. [En línea]. http://black-byte.com /tutorial/primeros-pasos-en-opengl/3/.

[8] Brescianni, Tommaso. 2008. *Master Thesis: Modelling, Identification and Control of a Quadrotor Helicopter*. 180 pags.

[9] Bresciani, Tommaso. 2008. Modelling, Identification and Control of a Quadrotor Helicopter.

[10] Brown, Thomas. 2001. *Handbook of operational amplifier applications*. [En línea]. http://www.ti.com/lit/an/sboa092a/sboa092a.pdf

[11]Brown, Ward. 2011. *Brushless DC Motors Control Made Easy*. [En línea]. http://ww1.microchip.com/downloads/en/AppNotes/00857a.pdf

[12] Budynas, Richard. 2008. *Diseño en ingeniería mecánica de Shigley*. 8va edición. Editorial Mc GrawHill. México, D.F. 1059ppt.

[13] Búrbano, Mario. 2011. *Implementación de una interfaz de datos inalámbrica RS-232 para el sensor laser de distancia Hokuyo URG-04LX-UG01*. Guatemala, Universidad del Valle de Guatemala.

[14] Colton, S. 2010. *The balance filter*. Massachusetts Institute of Technology. [En línea]. web.mit.edu/scolton/www/filter.pdf

[15] Fox, Robert. 1995. *Introducción a la mecánica de fluidos*. 4ta. Edición. Editorial McGraw Hill. México, D.F. 916ppt.

[16] Fritz, Walter. *Sistemas Inteligentes* [En línea] http://www.intelligent-systems.com.ar/intsyst/overviewSp.htm

[17] Globedia. 2011. *Parrot AR Drone*. http://gt.globedia.com/parrot-drone-cuadricoptero-wifi-iphone-ipad-cuesta-299-euros

[18] Gray, Robert. 2009. *Entropy and Information theory.* Estados Unidos, Stanford University. [En línea] http://ee.stanford.edu/~gray/it.pdf

[19] Gupta, A. 2009. *Microchip Technology Inc. AN1247, Communication Device Class (CDC) Host*.

[20] Hobby King. 2011. *Turnigy 2217 20turn 860kv 22A Outrunner*. [En línea]. http://www.hobbyking.com/hobbyking/store/uh_viewItem.asp?idProduct=5691

[21] Hobby King. 2011. *Turnigy 5000mAh 3S 20C Lipo Pack(DE Warehouse)*.[En línea] http://www.hobbyking.com/hobbycity/store/uh_viewItem.asp?idProduct=10355& Product_Name=Turnigy_5000mAh_3S_20C_Lipo_Pack_(DE_Warehouse)

[22] Hobby King. 2011. *Turnigy Plush 25amp Speed Controller*. [En línea]. http://www.hobbyking.com/hobbyking/store/uh_viewItem.asp?idProduct=2163

[23] Hokuyo Automatic Co., LTD. 2006. *Communication Protocol Specification for SCIP 2.0 Standard*. [En línea] http://www.hokuyo-aut.jp/02sensor/07scanner/download/data/ URG_SCIP20.pdf

[24] Hokuyo Automatic Co., LTD. 2009. *Scanning Laser Range Finder URG- 04LX-UG01 Specifications*. [En línea] http://www.hokuyo-aut.jp/02sensor/07scanner/download/data/ URG-04LX_UG01_spec.pdf

[25] Instructables - Make, How To, and DIY. 2010. *Accelerometer & Gyro Tutorial*. [En línea]. http://www.instructables.com/id/Accelerometer-Gyro- Tutorial/?ALLSTEPS

[26] Kets, Brecht. 2007. *Getting started with XNA*. [En línea]. http://www.3dgame programming.net/

[27] LAYADA. 2011. *XBee radios*. [En línea]. http://www.ladyada.net/make/xbee/ind ex.html

[28] López Ortega Alvaro. 2001. *Introducción a OpenGl*. [En línea]. http://www.alobbs .com/revistas/opengl1.

[29] Lu, Feng; Milios, Evangelos. *Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans.* Canadá, Universidad de York.

[30] Merriam – Webster Dictionary. *Autonomy*. [En línea] http://www.merriam-webster.com/ dictionary/autonomy

[31] Microchip. 2011. dsPIC30F4011. [En línea]. http://www.microchip.com/wwwproducts/ Devices.aspx?dDocName=en010337

[32] Microsoft. *C# language tour*. http://msdn.microsoft.com/en-us/library/67ef8sbd(v=VS.71).aspx

[33] Mijangos, Luis. 2005. *Diseño e Implementación de un Sistema de Exploración Robotizado y Autárquico.* Guatemala, Universidad del Valle de Guatemala.

[34] Miles, Rob. 2011. *Microsoft XNA Game Studio 4.0*. Washington, Estados Unidos. Microsoft Press. 465 págs.

[35] Mott, Robert. 2006. *Mecánica de fluidos*. 6ª Edisión. Editorial Pearson. México D.F. 644ppt.

[36] Muñoz, Miguel. 2010. *Principios básicos*. [En línea]. http://www.manualvuelo.com/PBV/PBV17.html

[37] Murray, Richard. 2011. Feedback *Systems: An Introduction for Scientists and Engineers*. [En línea]. http://www.cds.caltech.edu/~murray/books/AM08/pdf/am08-xferfcns_19Jul11.pdf

[38] Newman, P.; Clark, S. 2006. *A Solution to the Simultaneous Localization and Map Building (SLAM) Problem.* Australia, University of Sydney. 14 páginas.

[39] Palomino, Evelio. 2011. *Análisis de la eficiencia del sistema de aislamiento de vibraciones de grupos electrógenos*. [En línea]. http://scielo.sld.cu/scielo.php?pid=S1815-59442011000100004&script=sci_arttext

[40] Parallax. 2010. *XBee PRO 60mW Wire Antenna (XBP24-AWI-001)*. [En línea]. http://www.parallax.com/Store/Accessories/CommunicationRF/tabid/161/ProductID/641/List/0/Default.aspx?SortField=ProductName

[41] Pérez, Diego. 2006. *Sensores de distancia por ultrasonidos*. [En línea]. http://www.alcabot.com/alcabot/seminario2006/Trabajos/DiegoPerezDeDiego.pdf

[42] Pfister, Samuel; Roumeliotis, Stergios; Burdick, Joel. *Weighted Line Fitting Algorithms for Mobile Robot Map Building and Efficient Data Representation.* Estados Unidos, California Institute of Technology.

[43] Reed, Aaron. 2011. *Learning XNA 4.0*. Estados Unidos. O'Reilly Media, Inc. 540 págs.

[44] «25: Robotics» Russell, Stuart; Norvig, Peter. 2003. *Artificial intelligence – a modern approach*. 2da edición. Estados Unidos, Pearson Education, Inc. Páginas 902 – 942.

[45] «3: Problem-solving» Russell, Stuart; Norvig, Peter. 2003. *Artificial intelligence – a modern approach*.2da edición. Estados Unidos, Pearson Education, Inc. Páginas 59-136.

[46] Sánchez, Omar. *Odometría.* [En línea]. http://omarsanchez.net/Documents/Odometr%C3%ADa.pdf

[47] Saravia, Roberto. 2011. *Vuelo automatizado*. Guatemala, Universidad del Valle de Guatemala.

[48] Schuller, Daniel. 2011. *C# Game Programming For Serious Game Creation*. Boston, Estados Unidos. Course Technology. 443 págs.

[49] Seifert, K., & Camacho, O. 2008. *Implementing Positioning Algorithms Using Accelerometers*. [En línea]. www.freescale.com/files/sensors/doc/app_note/AN3397.pdf

[50] Smith, Julius. 2007. *Introduction to digital filters* [En línea] https://ccrma.stanford.edu/~jos/filters/filters.html

[51] Smith, William. 1998. *Fundamentos de la ciencia e ingeniería de materiales*. 3ra Edisión. Editorial McGraw Hill. Madrid, España. 715ppt.

[52] Solana, J. 2011. *¿Fibra de vidrio?*. [En línea]. www.jsolana.com.mx/fibra/

[53] Sparkfun electronics. 2009. *IMU Analog Combo Board Razor - 6DOF Ultra-Thin IMU*. [En línea]. www.sparkfun.com/products/10010

[54] Sparkfun electronics. *XBee/Xbee-PRO OEM RF Modules*. [En línea]. http://www.sparkfun. com/datasheets/Wireless/Zigbee/Xbee-Manual.pdf.

[55] Starlino Electronics. 2011. *A Guide to using IMU (Accelerometer and Gyroscope Devices) in Embedded Applications*. [En línea] http://www.starlino.com/imu_guide.html

[56] StatTrek. *Probability Distributions.* [En línea] http://stattrek.com/lesson2/ probabilitydistribution.aspx

[57] Sujan, Vivek; Dubowsky, Steven. 2005. *Efficient Information – based Visual RoboticMapping in Unstructured Environments.* Estados Unidos, Massachusetts Institute of Technology.

[58] Texas Instruments. 2008. *Accelometers and how they work*. [En línea]. www2.usfirst.org/2005comp/Manuals/Acceler1.pdf

[59] Texas instrument. 2011. *Motor Control – Brushless DC (BLDC) Motors*. [En línea]. http://www.ti.com/ww/en/motor_drive_and_control_solutions/motor_control_typ e_brushless_dc_BLDC.htm

[60] Thurn, Sebastian; Burgard, Wolfram; Fox, Dieter. 2000. *A Real-Time Algorithm for Mobile Robot Mapping With Applications to Multi-Robot and 3D Mapping*. Estados Unidos, Carnegie Mellon University. 8 páginas.

[61] Turnigy. 2009. *Manual for brushless motor speed controller*. [En línea]. http://www.hobbyking.com/hobbyking/store/uploads/809043064X351363X29.pdf

[62] UNC Physics Department. *Definitions of measurement Uncertainty terms.* [En línea] http://www.physics.unc.edu/~deardorf/uncertainty/definitions.html

[63] Universidad Nacional de Colombia. *Teoría de Sistemas.* [En línea] 2011. http://www.virtual.unal.edu.co/cursos/sedes/manizales/4060001/Contenido/CAPI TULO%202-%20Concepto%20de%20sistemas/Pages/Clasificacion.htm

[64] Vester, D. 2009. *Combine Gyroscope and Accelerometer Data*. [En línea]. http://stackoverflow.com/questions/1586658/combine-gyroscope-and- accelerometer-data

[65] Welch, G., & Bishop, G. 2010. *The Kalman Filter*. [En línea]. http://www.cs.unc.edu/~welch/kalman/

[66] Whittier, Bob. 2011. *Trabajando con fibras de vidrio*. [En línea]. http://www.mimecanicapopular.com/verhaga.php?n=123

[67] Yedamale, Padmaraja. 2011. *Brushless DC (BLDC) Motor Fundamentals*. [En línea]. http://ww1.microchip.com/downloads/en/AppNotes/00885a.pdf

[68] Zhong, Jinghua. 2006. *PID Controller Tuning: A Short Tutorial*. [En línea]. http://saba.kntu.ac.ir/eecd/pcl/download/PIDtutorial.pdf

[69] Zill, Dennis. 2007. *Ecuaciones diferenciales con aplicación al modelado*. 8va edición. Editorial Thomson. 393ppt.

XVI. APÉNDICE

A. ESTRUCTURA

FIGURA NO. 212. PLANO DE BRAZO DEL CUADCÓPTERO

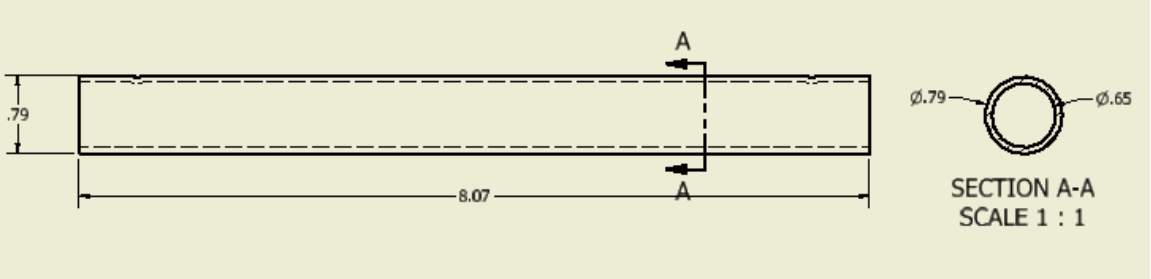


FIGURA NO. 213. PLANO DE CAJA CENTRAL DEL CUADCÓPTERO

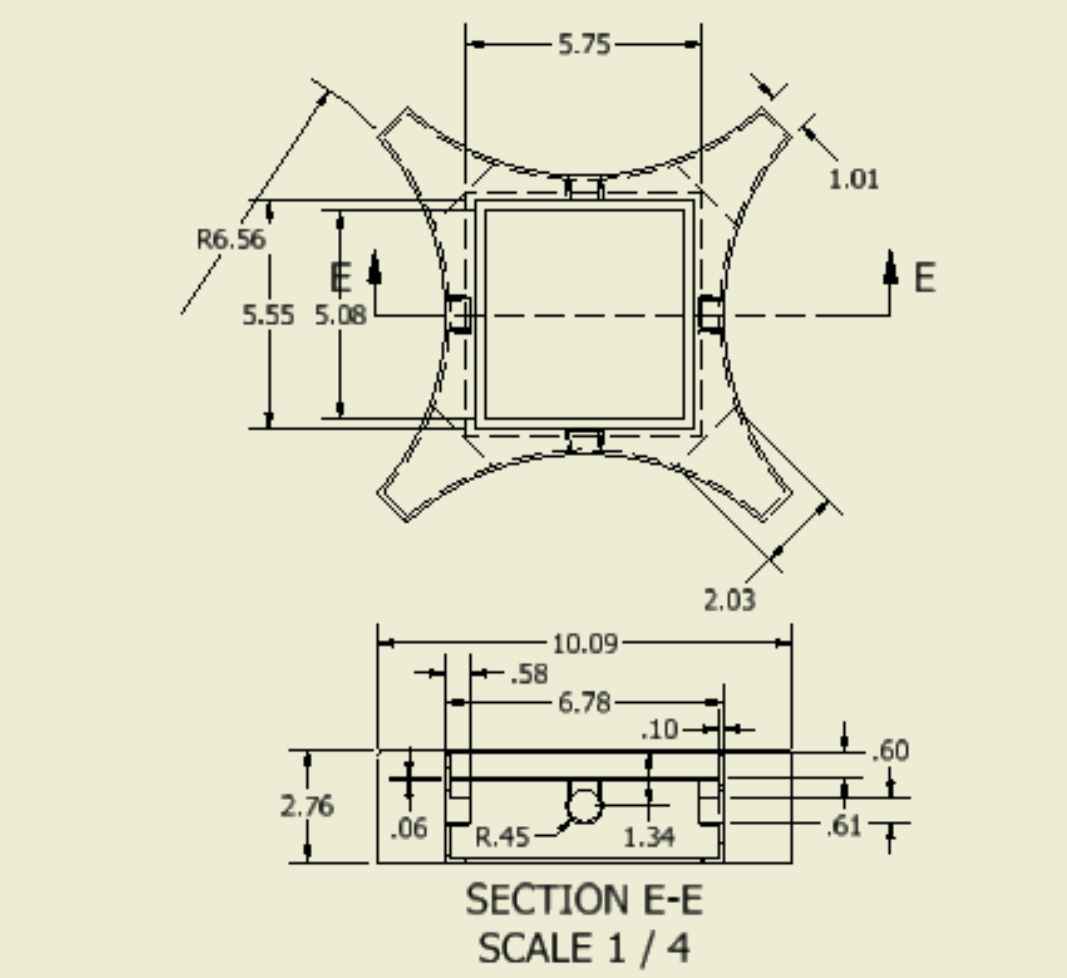


FIGURA NO. 214. PLANO DE LA CRUZ CENTRAL DEL CUADCÓPTERO

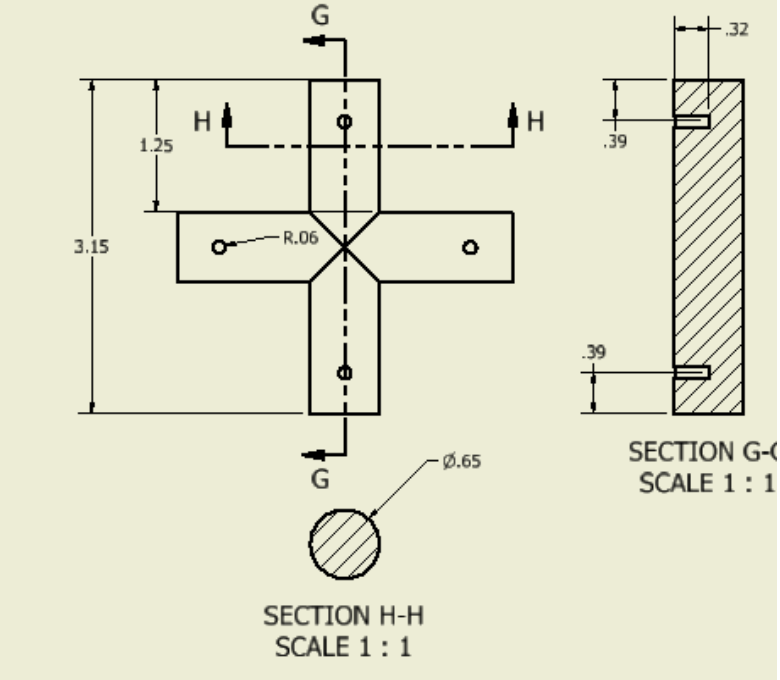


FIGURA NO. 215. PLANO DE PARACHOQUES DEL CUADCÓPTERO

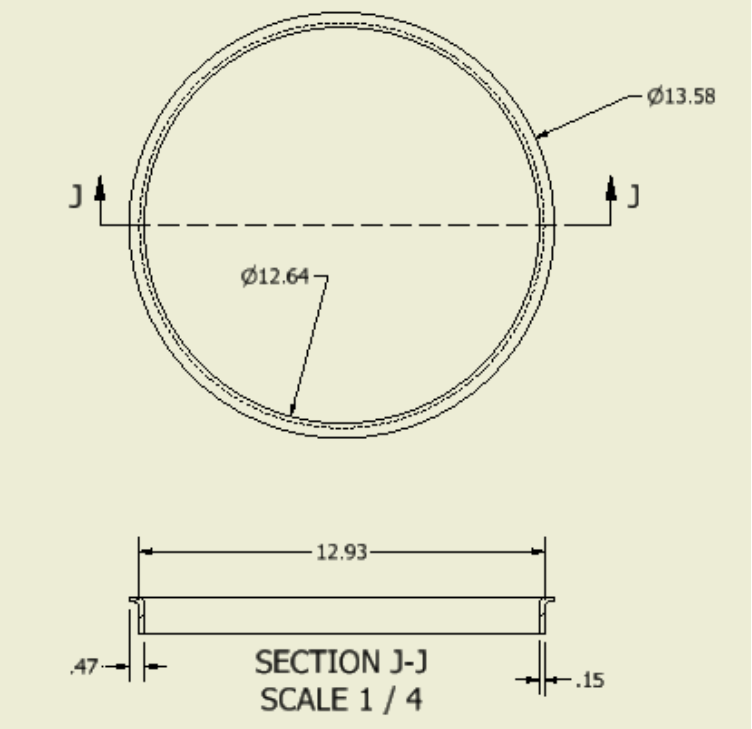


FIGURA NO. 216. PLANO DE LAS BANCADAS DEL CUADCÓPTERO

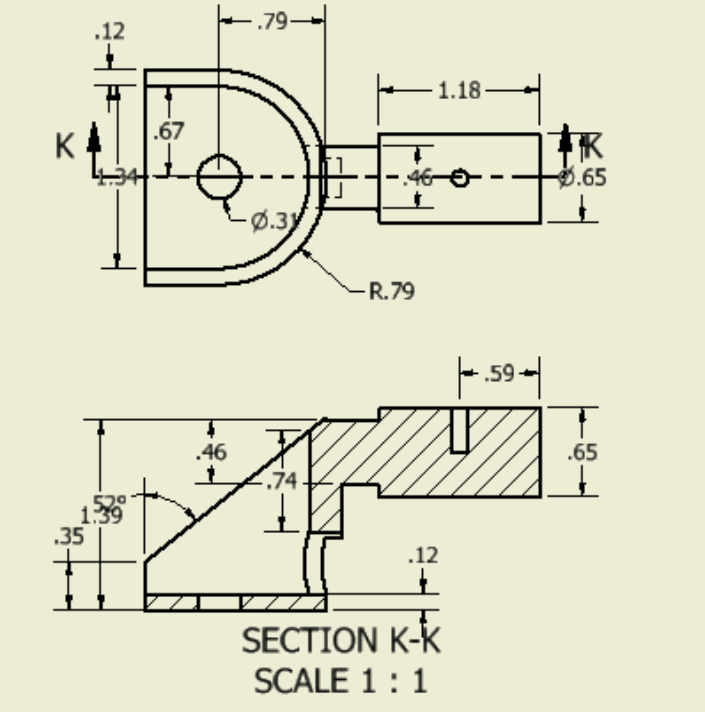
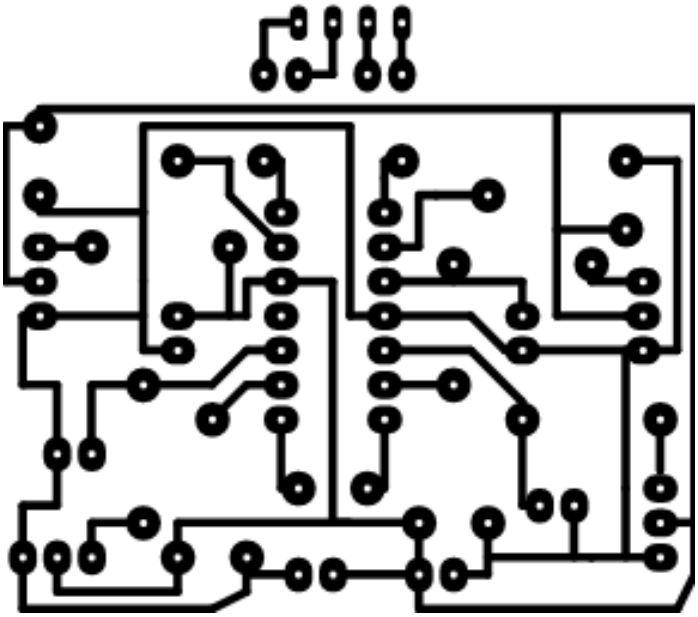


FIGURA NO. 217. PLACA PARA CIRCUITO DETECTOR DE OBJETOS



B. ESTABILIZACIÓN Y CONTROL

1. CÓDIGO PRINCIPAL DEL DSPIC MAIN.C

```
//-------------------------------------------------
//-------------------------------------------------
//              Control del Cuadcoptero
//-------------------------------------------------
//-------------------------------------------------



//------ Archivos incluidos ------------------
 #include <p30F4011.h>

 #include <UART2.h>
 #include <ADC.h>
 #include <PWM.h>
 #include <math.h>
 #include <angulo.h>
 //#include <float.h>

//------ manejo de bits -----------------------
 #define BIT_SET(VAR,BIT) (VAR |= (1<<BIT))
 #define BIT_CLEAR(VAR,BIT) (VAR &= ~(1<<BIT))
 #define BIT_TEST_SET(VAR,BIT)   if((VAR & (1<<BIT))!=0)
 #define BIT_TEST_CLEAR(VAR,BIT) if((VAR & (1<<BIT))==0)
 #define DELAY(VAR)   for ( i =0; i<VAR;i++){for ( a =0; a<1000;a++);} //delay

//------ palabras de configuracion -----------
_FOSC(CSW_FSCM_OFF & XT_PLL16);  //Set up for crystal multiplied by 16x PLL
_FWDT(WDT_OFF);                  //Turn off the Watch-Dog Timer.
_FBORPOR(MCLR_EN & PWRT_OFF);    //Enable MCLR reset pin and turn off the power-
up timers.
_FGS(CODE_PROT_OFF);             //Disable Code Protection

//------ variables generales -----------------
 int i;
 int a;
 int count1=0;
 char count=0;
 #define OneG   (signed int)tempu[2]
 #define ZeroG  (signed int)tempu[6]
 #define VentanaIa 40.0
//------ variables control -------------------
 unsigned int receive=0x30;
 char comandosControl=0;
 int velGiro=0.0;
 float setpoints [4] ={32.0,32.0,1.0,1.0} ;

//------ variables PID -----------------------
  float motor[5]={0,0,0,0,0};

  float errorP1=0;
  float errorI1=0;
  float errorD1=0;
  float errorP2=0;
  float errorI2=0;
  float errorD2=0;
```

```
  float errorDD1=0;
  float errorDD2=0;

  float errorP1_ant=0,errorP2_ant=0,errorD1_ant=0,errorD2_ant=0;

  float nivelGeneral;
  float PIDangulo[2];
  float angZ=0;

  float errorIaz=0.0;



//======================== Control PID ===================================

 void controlPID()
 {

    BIT_TEST_SET(comandosControl,2)     // 1=pruebas, 0 = pid
    {
      PWMSet(setpoints[3]/64,setpoints[3]/64,setpoints[3]/64,setpoints[3]/64);
    }
    else
    {
         if (setpoints[3]<5)      //apaga motores si el nivel es bajo
         {
            PWMSet(.25, .25, .25 , .25);
         }
         else
         {
// calcula el error del angulo
            errorP1 = 0.0174534*(setpoints[0]-32.0)*.08 - (int)(10000*Axz )/10000.0;
            errorP2 = 0.0174534*(setpoints[1]-32.0)*.08 - (int)(10000*Ayz )/10000.0;
// calcula el error Integrados
             errorI1 += errorP1;
             errorI2 += errorP2;
// calcula el error Derivativo
             errorD1 = errorD1*0.92 + (errorP1-errorP1_ant)*0.08;
             errorD2 = errorD2*0.92 + (errorP2-errorP2_ant)*0.08;
// calcula la segunda derivada
             errorDD1= errorDD1*0.92 + (errorD1-errorD1_ant)*0.08;
             errorDD2= errorDD2*0.92 + (errorD2-errorD2_ant)*0.08;

          PIDangulo[0]=0.3*3.60*errorP1  + 1.0*(6.0 )*errorD1 + 40.0*(31.0)*(errorDD1);
          PIDangulo[1]=0.3*3.60*errorP2  + 1.0*(13.0)*errorD2 + 40.0*(31.0)*(errorDD2);


// datos anteriores para calcular derivadas
             errorP1_ant=errorP1;
             errorP2_ant=errorP2;
             errorD1_ant=errorD1;
             errorD2_ant=errorD2;

//Saturaciones del PID para no generar sobreelevacion
             if (PIDangulo[0]>.075)
                PIDangulo[0]=.075;
             if (PIDangulo[0]<-.075)
                PIDangulo[0]=-.075;
             if (PIDangulo[1]>.075)
                PIDangulo[1]=.075;
             if (PIDangulo[1]<-.075)
                PIDangulo[1]=-.075;
```

```
//Saturacion del nivel de integral
             if (errorI1>VentanaIa)
                errorI1=VentanaIa;
             if (errorI1<-VentanaIa)
                errorI1=-VentanaIa;
             if (errorI2>VentanaIa)
                errorI2=VentanaIa;
             if (errorI2<-VentanaIa)
                errorI2=-VentanaIa;

             PIDangulo[0]+=0.00002*9.0*errorI1;
             PIDangulo[1]+=0.00002*9.0*errorI2;

// PI vel Angular en plano Z
             angZ=(float)velGiro*1.0-Wxy;
             errorIaz+=angZ*0.005;
             angZ=angZ*0.0005*(19.0)+errorIaz*0.000004*(13.0);


//calcula nivel general de alimentacion de los motores
             nivelGeneral=(setpoints[3]-5.0)/(63.0-5.0)*.25+.55;

//Aplica el PID y lo linealiza
             motor[1]= nivelGeneral+PIDangulo[0]+angZ;
             motor[2]= nivelGeneral+PIDangulo[1]-angZ;
             motor[3]= nivelGeneral-PIDangulo[0]+angZ;
             motor[4]= nivelGeneral-PIDangulo[1]-angZ;

// Aplica PWM
             PWMSet(motor[1],motor[2], motor[3] , motor[4]);

          }
     }
}



//----------------========== INTERRUPCION ================================
void __attribute__((__interrupt__, auto_psv)) _T1Interrupt(void){
     IFS0bits.T1IF = 0;

     tiempo+=0.0009;
     controlPID();                                        //llama al PID y
refresca PWM
     // do sonething
     return ;
 }

int main(void)
{
    //======================== Inicializa puertos ============================
     TRISF=0x0000;
     LATF=0x0000;
     PORTE = 0;
     PORTD = 0;
     TRISE = 0;
     TRISB = 0xff;

    //==================== INICIALIZA UART2 ==============================
     START_UART2();

    //==================== INICIALIZA ADC1 ==============================
     ADC_init();
```

```
    //====================== INICIALIZA PWM ================================
     PWM_Init();

    //================ INICIALIZA INTERRUPCION ==========================-
     IPC0   = IPC0 | 0x7000;          // Interrupt priority level = 7
     IFS0bits.T1IF = 0;               // Clear T1IF
     IEC0bits.T1IE = 1;               // Enable Timer1 interrupts
     T1CON = 0x8000;
     T1CONbits.TCKPS=1;
     PR1= 10000;

    //================== CONDICIONES INICIALES ============================

// promedia el voltaje de cero G
    for (count=0;count<32;count++)
      {

          DoMeasureXYZ();
          tempu[0]+=AccX;
          tempu[1]+=AccY;
          tempu[2]+=AccZ;
          tempu[3]+=GirX;
          tempu[4]+=GirY;
          tempu[5]+=GirZ;
          tempu[6]+=AccX+AccY;
          DELAY(5);
      }


// Zero en cada eje del imu
      tempu[0]=tempu[0]>>5;
      tempu[1]=tempu[1]>>5;
      tempu[2]=tempu[2]>>5;
       tempu[3]=tempu[3]>>5;
       tempu[4]=tempu[4]>>5;
      tempu[5]=tempu[5]>>5;
      tempu[6]=tempu[6]>>6;


// Guarda la sensibilidad del Accelerometro
      Sensitivity= (1.0/(float)(OneG-ZeroG));

       LATEbits.LATE8=0;

//============================ MAIN ======================================
     while(1)
     {
            DoMeasureXYZ();
            tiempo=0.0013;
            calculaAngulo();

// recepción de datos
            while(UART2_DATA_READY)
            {
                tiempo+=0.0001;
                receive=UART2_READ;
                if(receive<64&&receive>=0)
                    setpoints[0]=receive;
                if(receive<128&&receive>=64)
                    setpoints[1]=receive-64;
                if(receive<192&&receive>=128)
                {
```

```
                    setpoints[2]=receive-128;
                    comandosControl=receive-128;
                    BIT_TEST_SET(comandosControl,1)
                        velGiro=12;
                    else
                        velGiro=-12;
                    BIT_TEST_CLEAR(comandosControl,0)
                        velGiro=0;

//BIT_TEST_SET(comandosControl,3)
                      if ((comandosControl & 16) == 16 )
                    {
                        LATEbits.LATE8=0;
                        LATEbits.LATE8=1;
                        LATEbits.LATE8=0;
                    }


                }
                if(receive<255&&receive>=192)
                    setpoints[3]=(int)receive-192;
            }


            /*
            count++; // envia datos a la computadora
            if(count>30)
            {
                count=0;
                tiempo+=0.001;
                SEND(Axz*1800.0/3.1416*3.0);
                UART2_WRITE(44);
                SEND(Ayz*1800.0/3.1416*3.0);
                UART2_WRITE(44);
                UART2_WRITE(13);
                UART2_WRITE(10);
            }
            */
    }
    return(0);
}
```

2. CÓDIGO AUXILIAR ADC.H

```
#include <p30F4011.h>

#define Start_Sample() ADCON1bits.SAMP=1

#define ADC_Done  ADCON1bits.DONE

// inicializa el ADC a 500ksps, con voltajes de referencia en AVdd y AVss
#define ADC_init()          \
{                           \
    ADPCFG=0;               \
    ADCON1 = 0x00E0;        \
    ADCON2 = 0;             \
    ADCON3bits.SAMC=3;      \
    ADCON3bits.ADRC=0;      \
    ADCON3bits.ADCS=6;      \
    ADCON1bits.ADON=1;      \
}

int ADC_Read(char canal)
{
    ADCHSbits.CH0SA=canal;
    //ADCON1bits.ADON=1;
    Start_Sample();
    while (!ADCON1bits.DONE);
    ADCON1bits.DONE=0;
    return ADCBUF0;
}

signed int AccX, AccY, AccZ;
signed int GirX, GirY, GirZ;

void DoMeasureXYZ()
{
    AccX = ADC_Read(0);        // Measure X_out     acceleracion
    AccY = ADC_Read(1);        // Measure Y_out
    AccZ = ADC_Read(2);        // Measure Z_out
    GirX = ADC_Read(3);        // Measure X_out    vel angular
    GirY = ADC_Read(4);        // Measure Y_out
    GirZ = ADC_Read(5);        // Measure Z_out
    return ;
}
```

3. CÓDIGO AUXILIAR ANGULO.H

```
float RxAcc=0, RyAcc=0, RzAcc=0;
float RateXZ=0,  RateYZ=0;
float AxzGyro=0, AyzGyro=0;
float AxzAcc=0,  AyzAcc=0;
float ErrorAcc=0;
float wGyro=300.0;
float Axz=0.0,Ayz=0.0, Wxy=0.0;
float biasx=0;
float biasy=0;

float tiempo=0.002;
unsigned int tempu[7]={0,0,0,0,0,0,0};
float Sensitivity =0;

void calculaAngulo()
{
// convertir a G's
        RxAcc = (AccX - (signed int) tempu[0])*Sensitivity;
        RyAcc = (AccY - (signed int) tempu[1])*Sensitivity;
        RzAcc = (AccZ - (signed int) tempu[6]   )*Sensitivity;

// convertir a rad/s
        RateXZ=-(float)(GirX - (signed int)tempu[3])*0.0026331588*6.70;
        RateYZ=-(float)(GirY - (signed int)tempu[4])*0.0026331588*6.70;

// Obtiene el angulo segun el giroscopio (integrando)
        AxzGyro=Axz + (RateXZ-biasx)*tiempo ;
        AyzGyro=Ayz + (RateYZ-biasy)*tiempo ;

// Calcula el angulo segun acelerometro (trigonometria+ filtro pasa baja)
                    // cuidado con el desfase (retardo) a frecuencias bajas
        AxzAcc=AxzAcc*0.98+0.02*atan2(RxAcc,RzAcc);
        AyzAcc=AyzAcc*0.98+0.02*atan2(RyAcc,RzAcc);

//nivel de confianza en el giroscopio
        ErrorAcc=ErrorAcc*0.98 + (1.0-sqrt(RxAcc*RxAcc+RyAcc*RyAcc+RzAcc*RzAcc))*.02;
        wGyro=340.0+fabs(2500.0*ErrorAcc);

//ponderacion del Angulo real
        Axz=(AxzGyro*wGyro+AxzAcc)/(1+wGyro);
        Ayz=(AyzGyro*wGyro+AyzAcc)/(1+wGyro);

//quita el bias
        biasx = biasx*0.9991  + (-(AxzGyro-Axz)/tiempo + RateXZ)*0.0009 ;
        biasy = biasy*0.9991  + (-(AyzGyro-Ayz)/tiempo + RateYZ)*0.0009 ;

//filtro anguloZ
        Wxy=.99*Wxy +.01*(GirZ-(signed int)tempu[5]);
        return ;
}
```

4. CÓDIGO AUXILIAR PWM.H

```
#include <p30F4011.h>

// inicializa el pwm de control de motor
#define PWM_MC_Init();         \
    PTCONbits.PTEN   = 0;      \
    PWMCON1bits.PEN1H = 1;     \
    PWMCON1bits.PEN2H = 1;     \
    PWMCON1bits.PEN3H = 1;     \
    PTCONbits.PTCKPS = 1;      \
    PTCONbits.PTOPS  = 0;      \
    PTCONbits.PTSIDL = 0;      \
    PTCONbits.PTMOD  = 0;      \
    PTCONbits.PTEN   = 1;      \
    PTPERbits.PTPER = 10000;
// inicializa el pwm del modulo ccp
#define PWM_N_Init();           \
    PR2=39999;                  \
    OC1RS=10000;                \
    OC1R=10000;                 \
    OC1CONbits.OCM=6;           \
    OC1CONbits.OCTSEL=0;        \
    T2CONbits.TON=1;
// inicializa ambos pwm
#define PWM_Init();             \
    PWM_N_Init();               \
    PWM_MC_Init();

#define limiteSup 0.92
#define limiteInf 0.2
#define maxPWM 20000.0
#define maxPWM2 40000.0

// envia pwm a los motores
void PWMSet(float a, float b, float c , float di)
{

     if (a>limiteSup)
        a=limiteSup;
     if (b>limiteSup)
        b=limiteSup;
     if (c>limiteSup)
        c=limiteSup;
     if (di>limiteSup)
        di=limiteSup;

     if (a<limiteInf)
        a=limiteInf;
     if (b<limiteInf)
        b=limiteInf;
     if (c<limiteInf)
        c=limiteInf;
     if (di<limiteInf)
        di=limiteInf;

          PDC1=a*maxPWM;
          PDC2=b*maxPWM;
          PDC3=c*maxPWM;
          OC1RS=di*maxPWM2;
    return;}
```

5. CÓDIGO AUXILIAR UART2.H

```
#include <p30F4011.h>

#define START_UART2()\
U2BRG=0XA;      \
U2MODE=0x8000;  \
U2STA =0x8400;

#define UART2_DATA_READY U2STAbits.URXDA==1

#define UART2_READ (short)U2RXREG

#define UART2_WRITE(VAR)            \
    while(U2STAbits.UTXBF==1);  \
        U2TXREG=VAR;


//envía datos en ascii al uart2
int awr_ztemp;

void SEND( float var) {
        awr_ztemp= (int)var;
        if (var < 0) {
           UART2_WRITE('-');
           awr_ztemp = -awr_ztemp;
           }
        else
        {
            UART2_WRITE('+');
        }
        UART2_WRITE((awr_ztemp/100)%10+48);
        UART2_WRITE((awr_ztemp/10)%10+48);
        UART2_WRITE((awr_ztemp%10)+48);
}
```

6. MANUAL DEL CONTROLADOR DE MOTORES BRUSHLESS

TURNIGY Manual for Brushless Motor Speed Controller

Thank you for purchasing our Electronic Speed Controller (ESC). High power systems for RC model can be very dangerous; we strongly suggest you read this manual carefully. We have no control over the correct use, installation, application, or maintenance of our products, no liability shall be assumed nor accepted for any damages, losses or costs resulting from the use of the product. Any claims arising from the operating, failure or malfunctioning etc. will be denied. We assume no liability for personal injury, property damage or consequential loss resulting from our product or our workmanship. As far as is legally permitted, the obligation to compensation is limited to the invoice amount of the affected product.

Features:

- **Lithium battery Balance Discharge Monitoring and Protecting (BDMP) Design**, monitors in real time the discharge voltage of each lithium (Li-ion/Li-poly) cell in a battery pack. Don't worry about the over discharge problem again, your lithium battery pack will have a much longer life. **(Remark: This BDMP function is ONLY available for "SENTRY" series ESC)**
- Extreme low resistance, high current endurance.
- Full protection features: Low-voltage cutoff protection / Over-heat protection / Throttle signal lost protection
- 3 startup modes: Normal / Soft / Super-Soft, can be used for both fixed-wing aircraft or helicopter models
- Throttle range can be configured, fully compatible with all kinds of available transmitters.
- Smooth and accurate speed control, excellent throttle linearity.
- Microprocessor uses separate voltage regulator IC (except PULSAR-6A and PULSAR-10A), with high anti-jamming capability.
- Supports up to: 210000 RPM (2 poles), 70000 RPM (6 poles), 35000 RPM (12 poles) motors.
- The program card is a very small device which can be purchased additionally for easy programming the ESC in the field.
- With a program card, you can activate the music playing function of ESC, and there are 15 songs can be selected.

Specifications:

PLUSH Series											
Class	Model	Cont. Current	Burst Current (>10s)	BEC Mode	BEC Output	Battery Cell		User Programm-able	Balance Discharge Protection	Weight	Size
						Li-ion Li-poly	NiMH NiCd				L*W*H
6A	PLUSH-6	6A	8A	Linear	5V/0.8	2	5-6	Available	N/A	6g	24*12*6
10A	PLUSH-10	10A	12A	Linear	5V/2A	2-4	5-12	Available	N/A	9g	27*17*6
12A	PLUSH-12	12A	15A	Linear	5V/1A	2-4	5-12	Available	N/A	12g	32*24*8
	PLUSH-12E	12A	15A	Linear	5V/2A	2-4	5-12	Available	N/A	13g	32*24*10
18A	PLUSH-18	18A	22A	Linear	5V/2A	2-4	5-12	Available	N/A	19g	45*24*11
25A	PLUSH-25	25A	35A	Linear	5V/2A	2-4	5-12	Available	N/A	22g	45*24*11
	PLUSH-25-OPTO	25A	35A	N/A	N/A	2-4	5-12	Available	N/A	21g	45*24*11
30A	PLUSH-30	30A	40A	Linear	5V/2A	2-4	5-12	Available	N/A	25g	45*24*11
40A	PLUSH-40	40A	55A	Switch	5V/3A	2-6	5-18	Available	N/A	35g	55*28*12
	PLUSH-40-OPTO	40A	55A	N/A	N/A	2-6	5-18	Available	N/A	32g	55*28*11
60A	PLUSH-60	60A	80A	Switch	5V/3A	2-6	5-18	Available	N/A	60g	70*31*14
	PLUSH-60-OPTO	60A	80A	N/A	N/A	2-6	5-18	Available	N/A	56g	70*31*13
80A	PLUSH-80	80A	100A	Switch	5V/3A	2-6	5-18	Available	N/A	62g	70*31*14
	PLUSH-80-OPTO	80A	100A	N/A	N/A	2-6	5-18	Available	N/A	58g	70*31*13

SENTRY Series											
Class	Model	Cont. Current	Burst Current (>10s)	BEC Mode	BEC Output	Battery Cell		User Programm-able	Balance Discharge Protection	Weight	Size
						Li-ion Li-poly	NiMH NiCd				L*W*H
18A	SENTRY-18	18A	22A	Linear	5V/2A	2-4	5-12	Available	Available	24g	45*26*11
25A	SENTRY-25	25A	35A	Linear	5V/2A	2-4	5-12	Available	Available	27g	45*26*12
30A	SENTRY-30	30A	40A	Linear	5V/2A	2-4	5-12	Available	Available	29g	45*26*12
40A	SENTRY-40	40A	55A	Switch	5V/3A	2-6	5-18	Available	Available	40g	55*28*15
60A	SENTRY-60	60A	80A	Switch	5V/3A	2-6	5-18	Available	Available	65g	70*31*14
80A	SENTRY-80	80A	100A	Switch	5V/3A	2-6	5-18	Available	Available	67g	70*31*14

BASIC Series											
Class	Model	Cont. Current	Burst Current (>10s)	BEC Mode	BEC Output	Battery Cell		User Programm-able	Balance Discharge Protection	Weight	Size
						Li-ion Li-poly	NiMH NiCd				L*W*H
18A	BASIC-18	18A	22A	Linear	5V/2A	2-4	5-12	Available	Available	24g	45*26*11
25A	BASIC-25	25A	35A	Linear	5V/2A	2-4	5-12	Available	Available	27g	45*26*12

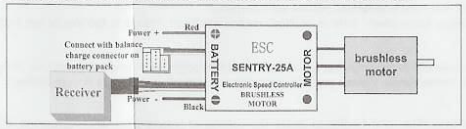
BEC Output Capability	Linear Mode BEC(5V/2A)				Switch Mode BEC(5V/3A)	
	2S Li-Poly	3S Li-Poly	4S Li-Poly	5S Li-Poly	2S — 4S Li-Poly	5S Li-Poly
Standard micro servos(Max.)	5	4	3	2	5	4

IMPORTANT! The ESC named "xxx-xxx-OPTO" hasn't a built-in BEC, so an UBEC (Ultimate-BEC) or an individual battery pack should be used to power the receiver. And an individual battery pack is needed to power the program card when setting the

TURNIGY Manual for Brushless Motor Speed Controller

programmable value of ESC, please read the user manual of program card for reference.

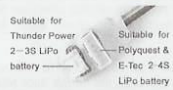
Wiring Diagram:



Lithium Battery Balance Discharge Monitoring and Protecting Adapter For "SENTRY" Series ESC:
We provide 2 kinds of Lithium Battery Balance Discharge Monitoring and Protecting Adapters for user to choose.

Adapter #1

Adapter #2



VERY IMPORTANT! You MUST connect the adapter with the balance charge connector on battery pack BEFORE connecting the main power to ESC. And if you use banana-shape connectors on main power wires (Input wires), please connect the black wire (negative polarity) BEFORE red wire (positive polarity). So the right sequence is:
Balance discharge adapter → BLACK wire of main power → RED wire of main power

Feature Explanation:

1. **Brake Settings**: Brake Enabled / Brake Disabled, default is Brake Disabled
2. **Battery Type**: Li-xx(Li-ion or Li-poly) / Ni-xx(NiMh or Nicd), default is Li-xx.
3. **Low Voltage Protection Mode(Cutoff Mode)**: Reduce / Cutoff Output Power, default is Reduce the output power gradually.
4. **Low Voltage Protection Threshold(Cutoff Threshold)**: Low / Medium / High, default is Medium.
   - **When NOT using balance discharge monitoring and protecting function** (i.e. **Do Not** plug the balance charge connector into the balance discharge protecting socket on ESC, in this case , the ESC only monitors the voltage of whole battery pack)
     1) For Li-xx battery, number of battery cells are calculated automatically, low / medium / high cutoff voltage for each cell are: 2.6V/2.85V/3.1V. For example: 3 cells Li-Poly, when medium cutoff voltage is set, the cutoff voltage is: 2.85*3=8.55V.
     2) For Ni-xx battery, low / medium / high cutoff voltages are 0%/45%/60% of the startup voltage (i.e. the initial voltage of battery pack), and 0% means low voltage cutoff function is disabled. For example: 10 cells NiMH battery, fully charged voltage is 1.44*10=14.4V, when "medium" cutoff voltage is set, the cutoff voltage is : 14.4*45%=6.5V.
   - **When using balance discharge monitoring and protecting function** (i.e. Plug the balance charge connector on battery pack into the balance discharge protecting socket on ESC, in this case, the ESC monitors not only the voltage of whole battery pack but also the voltage of each cell). For Li-xx battery, low / medium / high cut off voltage for each cell are: 2.6V/2.85V/3.1V. When the voltage of any cell in battery pack is lower than the cutoff threshold, the protecting program is activated.
5. **Startup Mode**: Normal /Soft /Super-soft, default is Normal startup.
   Normal is good for fixed-wing aircraft. Soft and Super-soft are good for helicopters. The initial speed of soft / super-soft mode is very slow, 1 second (soft startup) / 2 seconds (super-soft startup) from startup to full speed. But if throttle is closed (throttle stick is moved to bottom) and opened again (throttle stick is moved upwards) within 3 seconds after the first startup, the restart will be temporarily changed to normal mode to get rid of the chances of crash caused by slow throttle response in aerobatic fly.
6. **Timing**: Low / Medium / High, default is Low.
   In normal cases, low timing can be used for most motors. But for high efficiency, we recommend the **Low** timing for 2 poles motor and **Medium** timing for 6 poles and above. For higher speed, **High** timing can be chosen.
   Important! After you changing the timing setting, please test your RC model on ground firstly!

Special Hint
Some high KV out-runner motors have very special configuration, the space between each alnico is very large, and lots of ESCs can't drive these motors. After updating the program, our ESCs have very good compatibility with them. But some RC fans still have several questions about the programmable value for some special motors. So we just give some suggestions as follows:

Motor \ Programmable Value Suggestion	Timing	Startup Mode
General in-runner motor	Low	Usually, aircraft uses "normal" startup mode helicopter uses "super-soft" startup mode
General out-runner motor	Low or Medium	
Align 420LF (Made in TAIWAN, out-runner)	High (MUST)	
450TH (Made in TAIWAN, out-runner)	Low	Soft (MUST)

Begin To Use Your New ESC
Please start up the ESC in the following sequence:

1. Move the throttle stick to bottom position and then switch on the transmitter.
2. Connect battery pack to ESC, the ESC begins the self-test process, a special tone " ♪123" is emitted, which means the voltage of battery pack is in normal range, and then N "beep" tones will be emitted, means the quantity of lithium battery cells. Finally a long "beep------" tone will be emitted, which means self-test is OK, the aircraft/helicopter is ready to go flying.
   - If nothing is happened, please check the battery pack and all the connections;

- If a special tone " ♪ 5671♪" is emitted after 2 beep tones ("beep-beep-"), means the ESC has entered the program mode, i.e. the throttle channel of your transmitter is reversed, please set it correctly;
- If a very rapid "beep-beep-, beep-beep-" tone is emitted, means the input voltage is too low or too high, please check your battery's voltage.

3. "**VERY IMPORTANT!**" Because different transmitter has different throttle range, we strongly suggest you using the "Throttle Range Setting Function" to calibrate throttle range. Please read the instruction on page 4------"Throttle Range Setting".

Alert Tone

1. Input voltage abnormal alert tone: The ESC begins to check the voltage of battery pack when power on, if the voltage is not in acceptable range, such an alert tone will be emitted: "beep-beep-, beep-beep-,beep-beep-"(every "beep-beep-" has a time interval about 1 second.)
2. Throttle signal abnormal alert tone: When the ESC can't detect the normal throttle signal, such an alert tone will be emitted: "beep-, beep-, beep-". (Every "beep-" has a time interval about 2 seconds)
3. Throttle stick not at bottom position alert tone: When the throttle stick is not in bottom (lowest) position, a very rapid alert tone will be emitted: "beep-, beep-, beep-". (Every "beep-" has a time interval about 0.25 second.)

Protection Function

1. Start up protection: If the motor failed to start up in 2 seconds while the throttle stick is being moved upwards, the ESC will cut off the output power. In this case, the throttle stick **MUST** be moved to bottom again to restart the motor. (Such a situation happens in the following cases: The connection between ESC and motor is not reliable, propeller is blocked, gearbox is damaged, etc.)
2. Over-heat protection: When the temperature of ESC is higher than a factory-preset value, the ESC will reduce the output power.
3. Throttle signal lost protection: The ESC will reduce output power if throttle signal lost for 1 second, further lost for 2 seconds will cause its output to be cut off.

Program example

Setting startup mode to "super-soft", i.e. value #3 in program item #5

1. Enter Program Mode Switch on transmitter, move throttle stick to top, connect battery pack to ESC, wait for 2 seconds, "beep-beep" tone should be emitted. Then wait another 5 seconds, special tone like "5671♪" should be emitted, means program mode is entered.
2. Select Programmable Items Now you'll hear 8 tones in loop. When a long "beep------" tone is emitted, move throttle stick to bottom to enter the "Startup Mode"
3. Set Item Value (Programmable Value) "Beep-", wait for 3 seconds; "Beep-beep-", wait for another 3 seconds; then you'll hear "beep-beep-beep", move throttle stick to top, then a special tone "♪♪♪" is emitted, now you have set the "Startup Mode" item to the value of "Super-soft Startup"
4. Exit Program Mode After the special tone "♪♪♪", move throttle stick to bottom within 2 seconds.

Trouble Shooting

Trouble	Possible Reason	Action
After power on, motor can't work, no sound is emitted	The connection between battery pack and ESC is not OK	Check the power connection. Replace the connector.
After power on, motor can't work, such an alert tone is emitted: "beep-beep-, beep-beep-,beep-beep-" (Every "beep-beep-" has a time interval about 1 second)	Input voltage is abnormal, too high or too low	Check the voltage of battery pack
After power on, motor can't work, such an alert tone is emitted: "beep-, beep-, beep- "(Every "beep-" has a time interval about 2 seconds)	Throttle signal is abnormal	Check the receiver and transmitter Check the cable of throttle channel
After power on, motor can't work, such a alert tone is emitted: "beep-, beep-, beep-" (Every "beep-" has a time interval about 0.25 second)	Throttle stick is not in bottom(lowest) position	Move the throttle stick to bottom
After power on, motor can't work, a special tone "5671♪" is emitted after 2 beep tone (beep-beep-)	The direction of throttle channel is reversed, so the ESC has entered the program mode	Set the direction of throttle channel correctly
The motor runs in opposite direction	The connection between ESC and the motor need to be changed.	Swap any two wire connections between ESC and motor
The motor stop running while in working state	Throttle signal is lost	Check the receiver and transmitter Check the cable of throttle channel
	ESC has entered Low Voltage Protection mode	Land RC model as soon as possible, and then replace the battery pack
	Some Connections are not reliable	Check all the connections: battery pack connection, throttle signal cable, motor connections, etc.

TURNIGY Manual for Brushless Motor Speed Controller

Normal startup procedure:

Switch on transmitter, move throttle stick to bottom	Connect battery pack to ESC, special tone like "♪123" means power supply is OK	Several "beep-" tones should be emitted, presenting the quantity of lithium battery cells	When self-test is finished, a long "beep-----"tone should be emitted	Ready to go flying now

Throttle range setting: (Throttle range should be reset when a new transmitter is being used)

Switch on transmitter, move throttle stick to top	Connect battery pack to ESC, and wait for about 2 seconds	"Beep-beep-" tone should be emitted, means throttle range highest point has been correctly confirmed	Move throttle stick to the bottom, several "beep-" tones should be emitted, presenting the quantity of battery cells	A long "Beep-" tone should be emitted, means throttle range lowest point has been correctly confirmed

Program ESC with transmitter (4 Steps):

1. Enter program mode
2. Select programmable items
3. Set item value (Programmable value)
4. Exit program mode

1. Enter program mode

1) Switch on transmitter, move throttle stick to top position, connect the battery pack to ESC
2) Wait for 2 seconds, the motor should emit special tone like "beep-beep-"
3) Wait for another 5 seconds, special tone like " 5671♪" should be emitted, which means program mode is entered

2. Select programmable items:

After entering program mode, you can hear 8 tones in a loop in the following sequence. If you move the throttle stick to bottom within 3 seconds after one kind of tones, then this item will be selected.

1.	"beep"	brake	(1 short tone)
2.	"beep-beep-"	battery type	(2 short tone)
3.	"beep-beep-beep-"	cutoff mode	(3 short tone)
4.	"beep-beep-beep-beep-"	cutoff threshold	(4 short tone)
5.	"beep-----"	startup mode	(1 long tone)
6.	"beep-----beep-"	timing	(1 long 1 short)
7.	"beep-----beep-beep-"	set all to default	(1 long 2 short)
8.	"beep-----beep-----"	exit	(2 long tone)

Remark: 1 long "beep-----" = 5 short "beep-"

3. Set item value (Programmable value):

You will hear several tones in loop. Set the value matching to a tone by moving throttle stick to top when you hear the tone, then a special tone "♪♪♪" emits, means the value is set and saved. (Keeping the throttle stick at top position, you will go back to step 2 and select other items; Moving the stick to bottom within 2 seconds, you will exit the program mode directly)

Tones / Items	"beep-" 1 short tone	"beep-beep-" 2 short tones	"beep-beep-beep" 3 short tones
Brake	Off	On	
Battery type	Li-ion / Li-poly	NiMh / Nicd	
Cutoff mode	Reduce power	Shut down	
Cutoff threshold	Low	Medium	High
Startup mode	Normal	Soft	Super soft
Timing	Low	Medium	High

4. Exit program mode

There are 2 ways to exit program mode:

1. In step 3, after special tone "♪♪♪" , move throttle stick to bottom within 2 seconds.
2. In step 2, after tone "beep-----beep-----"(ie. The item #8), move throttle stick to bottom within 3 seconds.

C. MAPEO Y EXPLORACIÓN

1. SOLUCIÓN EN FORMA CERRADA PARA ALGORITMOS DE COINCIDENCIA DE PUNTOS [23]

A continuación se plantea la solución de mínimos cuadrados en forma cerrada de la función de distancia propuesta para los algoritmos de alineamiento y coincidencia.

Para n parejas de puntos: $P(x_i, y_i), P'(x'_i, y'_i)$ con $i = 1, \dots, n$ la función de distancia entre las P transformadas y P′ se define como

$$E_{dist}(\omega, T) = \sum_{i=1}^{n} |R_\omega P_i + T - P'_i|^2$$

$$= \sum_{i=1}^{n} \left((x_i cos\omega - y_i sin\omega + T_x - x'_i)^2 + (x_i sin\omega + y_i cos\omega + T_y - y'_i)^2 \right)$$

Al minimizar E_{dist}, se puede obtener una solución en forma cerrada para T_x, T_y y ω, de la siguiente forma:

$$\omega = \arctan \frac{S_{xy'} - S_{yx'}}{S_{xx'} - S_{yy'}}$$

$$T_x = \bar{x}' - (\bar{x} cos\omega - \bar{y} sin\omega)$$

$$T_y = \bar{y}' - (\bar{x} sins\omega - \bar{y} cos\omega)$$

En donde:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \,, \qquad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \,, \qquad \bar{x}' = \frac{1}{n}\sum_{i=1}^{n} x'_i \,, \qquad \bar{y}' = \frac{1}{n}\sum_{i=1}^{n} y'_i$$

$$S_{xx'} = \sum_{i=1}^{n} (x_i - \bar{x})(x'_i - \bar{x}') \,, \qquad S_{yy'} = \sum_{i=1}^{n} (y_i - \bar{y})(y'_i - \bar{y}')$$

$$S_{xy'} = \sum_{i=1}^{n} (x_i - \bar{x})(y'_i - \bar{y}') \,, \qquad S_{yx'} = \sum_{i=1}^{n} (y_i - \bar{y})(x'_i - \bar{x}')$$

2. CRITERIO DE EXPLORACIÓN BASADO EN ENTROPÍA RELATIVA [2]

Considere un vector X de variables aleatorias que contienen la posición de los puntos de un mapa. La entropía relativa provee una manera de extender el concepto de entropía al caso en que X es continuo.

El primer concepto necesario es el de la función de distribución de probabilidad (pdf por sus siglas en inglés) no informativa. La pdf no informativa $q_0(X)$ se define de manera que la probabilidad de que X pertenezca a un set $\mathcal{X}$ de valores solo depende del volumen $V(\mathcal{X})$ del set $\mathcal{X}$, es decir

$$\int_{\mathcal{X}} q_0(X)dX = V(\mathcal{X})$$

Se asume que $\mathrm{q}_0(\mathrm{X})$ es uniforme.

El contenido de información relativo a una pdf genérica $q(X)$ en X se define como la entropía relativa entre $q(X)$ y la pdf no informativa $q_0(X)$:

$$H_q = -\int q(X)\ln\frac{q(X)}{q_0(X)}dX$$

Se supone que todas las variables aleatorias involucradas están normalmente distribuidas.

Dada la información actual sobre el mapa, descrita por una distribución *a priori* $q(X)$, y dada la distribución *a posteriori* $q'(X)$ después de la medición del sensor, el incremento esperado en el contenido de información se puede calcular explotando la entropía relativa:

$$E[\Delta H] = E\left[H_{q'}\right] - H_q = -\iint \frac{q'(X|Z)}{q_0(X)}$$

$$= -\iint q'(X|Z)ln\frac{q'(X|Z)}{q_0(X)}dXdZ + \int q(X)ln\frac{q(X)}{q_0(X)}dX \qquad (1)$$

En donde Z es un vector de variables aleatorias que representan la posición de los puntos medidos. Este operador de expectativa es necesario pues el valor de $H_{q'}$ depende del resultado de la medición Z, la cual es una incógnita *a priori.*

Ahora se debe especificar la expresión de incertidumbre del criterio para los casos:

1. La posición de observación del robot no se ve afectada por incertidumbre (posición inicial). Si todas las variables aleatorias tienen una distribución normal, la expresión (1) es proporcional a $\ln\frac{\sigma}{\sigma_0} - \ln\frac{\sigma_{p,i}}{\sigma_0}$ para cada punto (en donde ${\sigma_0}^2$ es la varianza de q_0 y ${\sigma_{p,i}}^2$ es la varianza de $q(X)$ en el punto i). Sumando para todos los puntos detectados por el sensor, se tiene

$$E[\Delta H] \approx \sum_i \ln\frac{\sigma}{\sigma_{p,i}}$$

   Para la posición inicial del robot se tiene

$$E[\Delta H] = \sum_{i\in\mathcal{N}} \ln\frac{\sigma}{P} = N\ln\frac{\sigma}{P}$$

   En donde $\mathcal{N}$ es el set de puntos nuevos detectados por el sensor, $N = |\mathcal{N}|$ es el número de puntos nuevos, P es el perímetro máximo del ambiente a explorar, y σ^2 es la varianza del error de medición.

2. La posición del robot se ve afectada por incertidumbre. Sea $\mathcal{A}$ el set de puntos del mapa que el robot va a ver desde el punto de observación, la incertidumbre de la posición del robot afecta tanto a los puntos ya detectados como a los nuevos. La contribución de error debido al error de incertidumbre de posición del robot en distintos puntos medidos está correlacionada. En específico ${\sigma_{unc,i}}^2 = J_i \Lambda J_i^T$, en donde J_i es la matriz Jacobiana de la observación z_i del punto i, y Λ es la matriz de covarianza de los parámetros de la posición del robot $[x\ y\ \theta]$. El criterio se puede formular explotando la aditividad de la entropía relativa. Se supone que X se puede subdividir en subconjuntos disjuntos: Entonces la entropía de la variable está dada por la suma de dos contribuciones. La primera está

dada por la entropía $-\sum_j P_j \ln P_j$ asociado a los diferentes subconjuntos, en donde P_j es la probabilidad de que el valor de la variable considerada pertenezca al j-ésimo subconjunto. La segunda contribución está dada por la suma ponderada de las entropías en cada subconjunto. Entonces

$$H_q = -\int q(X) \ln \frac{q(X)}{q_0(X)} dX$$
$$= -\sum_j P_j \ln P_j + \sum_j P_j \left(\int q(X|j) \ln \frac{q(X|j)}{q_0(X)} dX \right)$$

Asumiendo que $\sigma_{unc,i} \gg \sigma$, y subdividiendo X en subconjuntos con un rango de σ, se puede aproximar la expresión anterior a

$$E[\Delta H] \approx \frac{1}{N+A} \sum_{i \in \mathcal{A} \cup \mathcal{N}} ln \frac{\sigma_{unc,i}}{\sigma} + N \, ln \frac{\sigma}{P} + \sum_{i \in \mathcal{A}} ln \frac{\sigma}{\sigma_{p,i}}$$

En donde $\sigma_{p,i}$ es la desviación estándar previa en el punto ya detectado i.

3. CÓDIGO DE SIMULACIÓN

Lenguaje de programación: C#

```
/*
 * Autor: Roberto Enrique Saravia Fernández
 * Archivo: Form1.cs
 * Descripción: Simulación de ambiente de exploración, y aplicación de
 *              algoritmos desarrollados
 *
 * */

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Resources;
using System.Collections;
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        private Bitmap myBitmap, myBitmap2; // bitmaps para representación gráfica
        Pen pen = new Pen(Color.Red);
        Pen pen1 = new Pen(Color.White);    // herramientas representación gráfica
        int control = 0;                    // control de inicio y movimientos
        Rectangle r1;                       //
        Random ran1 = new Random();         // generador de posición aleatoria
        int x0 = 0, y0 = 0, x1 = 0, y1 = 0; // (x0,y0) posición anterior,
                                            // (x1, y1) posición nueva
        int x1_r = 0, y1_r=0;               // posición con ruido aleatorio
        int ang_r = 0;                      // ruido de rotación
        Random ran = new Random();          // ruido o error en movimiento
        // mapa global creado
        float[,] mapa_polar = new float[683,2]; // coordenadas polares
                                            //[a,0] es r - [a,1] es angulo en grados
        float[,] mapa_xy = new float[683, 2];   // coordenadas cartesianas relativas
                                            //a posicion actual (x0, y0)
                                            //- [a,0] es x - [a,1] es y

        // variables método de escaneo
        float[] dist = new float[683];          // Coordenadas polares de escaneado
        float[,] dist_xy = new float[683, 2];   // Coordenadas cartesianas
                                            //relativas a posición de robot
        double angulo;
        int longi;
        double xs_, ys_;                        // variables de simulación de escaneo

        // variables método de proyección
        float[,] proy_xy = new float[683, 2];   // coordenadas cartesianas por punto proyectado
                                            // respecto a nueva posición
        int[] dist_n = new int[683];
        float[,] proy_pol = new float[683, 2];  // coordenadas polares por punto proyectado,
        ArrayList indexProy = new ArrayList();  // índice en proyección de cada punto proyectado
        ArrayList indexGlob = new ArrayList();  // índice en mapa global de cada punto proyectado

        // variables método de alineamineto y coincidencia
        int[] P1 = new int[683];            //puntos correspondientes por distancia mínima
                                            // (almacena índices)
        float[,] P2 = new float[683, 2];    //arreglo de puntos correspondientes para matching
                                            // range-point rule
```

```
        double w1 = 0.0;                    //rotación obtenida de regla de punto más cercano
        double Tx1 = 0.0, Ty1 = 0.0;        //traslación obtenida de regla del punto más cercano
        double w2 = 0.0;                    //rotación obtenida de regla matching range-point
        double Tx2 = 0.0, Ty2 = 0.0;     //traslación obtenida de regla matching range-point
        ArrayList dist_m = new ArrayList();
        int Br = 150/10;                            // Límite de radio
        int Bw = 10;                                // Límite de rango para sector de búsqueda

        // variables estrategia de exploración basada en información
        double acc = 0.01;                          // precisión del sensor
        int c = 0;              // distancia para alcanzar un candidato de siguiente posición
        float[,] candidatos;    //coordenadas x y y con respecto al punto actual
        int r = 150;            //Alcance máximo de sensor
        float nextPx = 0;
        float nextPy = 0;       //coordenadas de siguiente mejor posición

        //constructor
        public Form1()
        {
            InitializeComponent();                // se inicializan componentes gráficos
            pictureBox1.BackgroundImage = (Bitmap)Properties.Resources._2;  // ambiente como fondo
            myBitmap = new Bitmap(pictureBox1.Size.Width,
                pictureBox1.Size.Height,
                System.Drawing.Imaging.PixelFormat.Format24bppRgb);
            myBitmap.MakeTransparent();                  // crea bitmap de ambiente (transparente)
            myBitmap2 = new Bitmap(pictureBox2.Size.Width,
                pictureBox2.Size.Height,
                System.Drawing.Imaging.PixelFormat.Format24bppRgb);
            myBitmap2.MakeTransparent();            // crea bitmap de mapa generado (transparente)
            r1 = new Rectangle(0, 0, pictureBox1.Size.Width, pictureBox1.Size.Height);
        }


        private void pictureBox1_MouseClick(object sender, MouseEventArgs e)
        {
            //para realizar mapeo se debe de haber seleccionado alguna tecnica de exploración
            if (tecnicas.SelectedIndex != -1)
            {
                if (control == 0)
                {
                    pictureBox1.DrawToBitmap(myBitmap, r1);         // se copia en bitmap el fondo
                    Graphics graphicsObj;
                    graphicsObj = Graphics.FromImage(myBitmap);     // se crea objeto para dibujar

    // se genera posición aleatoria de inicio de robot, rechaza posiciones que atraviesen paredes
                    //    do
                    //    {
                    //        x0 = ran1.Next(1, 100) * pictureBox1.Size.Width / 100;
                    //        y0 = ran1.Next(1, 100) * pictureBox1.Size.Height / 100;
                    //    } while (compareColor(myBitmap.GetPixel(x0, y0), Color.Black));

                   //con motivos de facilidad de experimentación, se permite a usuario elegir la
                   //posición inicial del robot
                    x0 = e.X;
                    y0 = e.Y;                     // eventos del puntero para identificar posición

                    graphicsObj.DrawEllipse(pen, x0 - 2, y0 - 2, 4, 4);
                    graphicsObj.DrawImage(myBitmap, 0, 0, myBitmap.Width, myBitmap.Height);
                    graphicsObj.Dispose();
                    pictureBox1.Image = (Image)myBitmap;  // dibuja robot en su posición de inicio

                    scan();                     // se realiza escaneo en posición actual del robot

                    //se almacena primer segmento de mapa, correspondiente a primera escaneada
                    for (int i = 0; i < 683; i++)
                    {
                        mapa_polar[i, 0] = dist[i];
                        mapa_polar[i, 1] = (float)((i * 240.0) / 683);
                        mapa_xy[i, 0] = dist_xy[i, 0];
```

```
                    mapa_xy[i, 1] = dist_xy[i, 1];
                }

                // y se dibuja mapa local de escaneado
                //localMap();
                //y se dibuja mapa global de escaneado
                //plotGlobal();
                control = 1;                            //Se cambia a estado de exploración
            }
            else
            {
                //  se genera siguiente posición

                if (tecnicas.SelectedIndex == 0)    // si se está utilizando la técnica random
                {
                    //se generan valores teóricos de nueva posición
                 // do
                 // {
                 //       x1 = ran1.Next(0, 100) * pictureBox1.Size.Width / 100;
                 //       y1 = ran1.Next(0, 100) * pictureBox1.Size.Height / 100;
                 //   } while (compareColor(myBitmap.GetPixel(x1, y1), Color.Black) ||
                 //     !check_route(x0, y0, x1, y1) || (int)distancia2(x1, y1, x0, y0) > 75);

                    x1= e.X;
                    y1 = e.Y;                   //se generan por el usuario (experimentación)

                }
                if (tecnicas.SelectedIndex == 1) // si seleccionó la técnica del algoritmo
                {
                    algoritmo_AC(ang_r);        // llamada a método de algoritmo
                    x1 = x0 + (int)nextPx;
                    y1 = y0 - (int)nextPy;      // genera siguiente posición x1,y1 teórica

                    //Se revisa máximo permisible
                     while (compareColor(myBitmap.GetPixel(x1, y1), Color.Black))
                    {
                        float dist_r = (float)distancia2(x1, y1, x0, y0); //distancia
                        float dist_ang = (float)(angle(x0, y0, x1, y1) * 180 / Math.PI);
                                                                  //ángulo en grados
                        dist_r = dist_r - 4;
                        x1 = x0 + (int)(dist_r * Math.Cos((dist_ang*Math.PI)/180));
                        y1 = y0 - (int)(dist_r* Math.Sin((dist_ang* Math.PI) / 180));
                    }
                }

                Graphics graphicsObj;
                graphicsObj = Graphics.FromImage(myBitmap);     // objeto para gráficos

        //se añade ruido a nuevas coordenadas y ángulo de rotación para simular movimiento real
                x1_r = x1 + (ran.Next(10) - 4);
                y1_r = y1 + (ran.Next(10) - 4);

                graphicsObj.DrawEllipse(pen, x1_r - 2, y1_r - 2, 4, 4); //robot nueva posición
                graphicsObj.DrawEllipse(pen1, x0 - 2, y0 - 2, 4, 4);   // borra anterior
                graphicsObj.DrawLine(pen, x0, y0, x1_r, y1_r);         //traza recorrido

                myBitmap2 = new Bitmap(pictureBox2.Size.Width,
                 pictureBox2.Size.Height,
                System.Drawing.Imaging.PixelFormat.Format24bppRgb);
                myBitmap2.MakeTransparent();
                pictureBox2.BackgroundImage = (Image)myBitmap2;   // Se limpia gráfica

                traslate_map(x1, y1);      // se traslada mapa completo al nuevo punto teórico
                                           // con algoritmo de cambio de coordenadas
                //y se realiza proyección de mapa a la nueva posición con coordenadas teóricas
                proyeccion();

                // plotGlobal();                       // y se dibuja mapa global de escaneado
                // plotProy();                         // se dibuja proyeccion
```

```
                //actualiza posiciones
                x0 = x1_r;
                y0 = y1_r;
                ang_r = ran.Next(10);

                graphicsObj.DrawImage(myBitmap, 0, 0, myBitmap.Width, myBitmap.Height);
                graphicsObj.Dispose();
                pictureBox1.Image = (Image)myBitmap;    //Se actualiza la imagen en interfaz

                scan();                                 //realiza escaneo de nueva posición
                //localMap();

                //se realiza alineamiento de nuevo escaneado con respecto a la proyeccion
                //(mov. esperado)
                //y se encuentra la traslación y rotación real del movimiento.
                int count_a= 20;                    //número de iteraciones del algoritmo

                monitoreo4.Text = "";               //resetea monitoreo de variables

                do
                {                                   // inicia ciclo
                   point_correspondence_matching();      // se aplica regla de correspondencia
                   if(double.IsNaN(w1) || double.IsNaN(Tx1) || double.IsNaN(Tx2))
                       apply_transform(0,0,0);   // si retorna valor inválido, aplica
                                                 //transformación teórica
                   else
                       apply_transform(w1, Tx1, Ty1); // se aplica transformación w, Tx, Ty a
                                                      // mapa global si existe

                   proyeccion();                 // se genera proyección de mapa transformado
                   //plotGlobal();
                   //plotProy();
                    count_a--;
                } while (count_a != 0);             // 20 iteraciones

                point_correspondence_matching();    // vuelve a aplicar algoritmo para obtener
                                                    // parejas de puntos correspondientes
                incrementMap();                     // se unen mapas usando algoritmo de unión
                                                    //y eliminación de repeticiones

                plotGlobal();                      //se dibuja mapa global actualizado

            }

            // dibuja imagen nueva como fondo
            pictureBox1.DrawToBitmap(myBitmap, r1);
            pictureBox1.BackgroundImage = (Image)myBitmap;
        }
    }

    //MÉTODOS AUXILIARES DE SIMULACIÓN
    /*-------------------------------------------------------------------------------------*/

    //metodo para comparar color de pixel con otro color de referencia
    private bool compareColor(Color input, Color comp)
    {
        if (input.A.Equals(comp.A) && input.R.Equals(comp.R)&& input.G.Equals(comp.G)
            && input.B.Equals(comp.B))
            return true;
        else
            return false;
    }

    //calcula distancia entre posicion actual y siguiente posicion
    private double distancia2(double x_o, double y_o, double x_f, double y_f)
    {
        double dist2 = 0;
        dist2 = Math.Sqrt(Math.Pow((x_f - x_o), 2) + Math.Pow((y_f - y_o), 2));
```

```
        return dist2;
    }

    //determina el limite de la interseccion de sectores angulares-
    private float int_angular(float ang_max, float ang_min, float ang_p)
    {
        float intercept = 0;
        float dist_max = 0, dist_min = 0;


        //se calcula su distancia con el angulo maximo
        dist_max = Math.Abs(ang_p-ang_max);
        //se calcula distancia con angulo minimo
        dist_max = Math.Abs(ang_min - ang_p);

        //se revisa la distancia menor y esta determina el limite del sector
        if (dist_max <= dist_min)
            intercept = ang_max;
        else
            intercept = ang_min;

        return intercept;
    }

    //método para calcular angulo entre dos puntos
    public double angle(double px1, double py1, double px2, double py2)
    {
        //se obtiene dif. en x y y entre los puntos
        double pxRes = px2 - px1;
        double pyRes = py2 - py1;
        double angle = 0.0;

        // se calula el angulo
        if (pxRes == 0.0)
        {
           if (pyRes == 0.0)
                 angle = 0.0;
           else if (pyRes > 0.0)
                angle = System.Math.PI / 2.0;
           else
                angle = System.Math.PI * 3.0 / 2.0;
        }
        else if (pyRes == 0.0)
        {
           if (pxRes > 0.0)
                 angle = 0.0;
           else
                 angle = System.Math.PI;
        }
        else
        {
           if (pxRes < 0.0)
                 angle = System.Math.Atan(pyRes / pxRes) + System.Math.PI;
           else if (pyRes < 0.0) angle = System.Math.Atan(pyRes / pxRes) +
                    (2 * System.Math.PI);
           else
                 angle = System.Math.Atan(pyRes / pxRes);
        }
        //se devuelve el valor calculado
        return angle;
    }

    //método para revisar que trayectoria no atraviese objetos
    private bool check_route(int x_a, int y_a, int x_b, int y_b)
    {
        bool checkk = true;
        //primero se obtiene ángulo de siguiente posición respecto a 0 de robot
        double ang = 0.0;
```

```
            int posi = 0;
            double posi_ = 0.0;
            ang = angle(x_a, y_a, x_b, y_b);
             //se convierte ángulo a grados
            ang = ang * 180 / Math.PI;

            //si el ángulo esta fuera del alcance del sensor
            if (ang >= 240)
            {
                checkk = false;
            }
            else
            {
                //se encuentra el dato que corresponde a ese angulo
                posi_ = (683.0 * ang) / 240.0;
                posi = (int)posi_;

        //luego se verifica mapa local (arreglo dist) para verificar que no haya muro en el camino)
                 //si la distancia es menor al valor maximo del sensor, significa que hay muro
                if ((dist[posi] < 150 || (int)distancia2(x_a, y_a, x_b, y_b) > dist[posi])
                        && dist[posi] != -1)
                    checkk = false;
            }

            return checkk;
        }

        //verifica camino a puntos propuestos como mejor posición
        private bool check_path(float xpat, float ypat)
        {
            bool checkk = true;
            //primero se obtiene angulo de siguiente posicion respecto a 0 de robot
            double ang = 0.0;
            int posi = 0;
            double posi_ = 0.0;
            ang = angle(x0, y0, (x0+xpat), (y0+ypat));
            //se convierte angulo a grados
            ang = ang * 180 / Math.PI;

            //si el angulo esta fuera del alcanze del sensor
            if (ang >= 240)
            {
                checkk = false;
            }
            else
            {
                //se encuentra el dato que corresponde a ese angulo
                posi_ = (683.0 * ang) / 240.0;
                posi = (int)posi_;

                //luego se verifica mapa local (arreglo dist) para verificar
                // que no haya muro en el camino)
                //si la distancia es menor al valor maximo del sensor, significa que hay muro
                if ((dist[posi] <= (r/2)) && dist[posi] != -1)
                    checkk = false;
            }

            return checkk;
        }



      //MÉTODOS ESPECÍFICOS MÓDULO VUELO AUTÓNOMO
    /*-------------------------------------------------------------------------------------------*/

        //método para realizar escaneo de posición actual
        private void scan()
        {
            for (int i = 0; i < 683; i++)
```

```
            {
                //genera ángulo para realizar escaneo
                angulo = (i * (240.0 / 683) + ang_r) * Math.PI / 180;
                longi = 0;

                //obtiene coordenadas cartesianas, repite hasta que detecte pared
                // o llege a alcanze máximo de sensor
                do
                {
                    longi++;
                    xs_ = x0 + longi * Math.Cos(angulo);
                    ys_ = y0 - longi * Math.Sin(angulo);    // resta porque eje y esta invertido
                } while (!(compareColor(myBitmap.GetPixel((int)xs_, (int)ys_), Color.Black)
                    || longi > 150));

                dist[i] = longi;
                if (longi >= 150)
                {
                    //se le asigna valor como bandera para indicar que este punto esta vacío
                    dist[i] = -1;
                }
                else
                {
                    //x, y ambos en plano normal (y+ arriba)
                    dist_xy[i, 0] = (float)(longi * Math.Cos((i * (240.0 / 683) * Math.PI)
                                        / 180));
                    dist_xy[i, 1] = (float)(longi * Math.Sin((i * (240.0 / 683) * Math.PI)
                                        /180));
                }
            }
        }

     //usa proyección de escaneado anterior para alineamiento de nuevo escaneado y mapa acumulado
        private void proyeccion()
        {
            //Se inicializan arreglos de proyección con valores de puntos inválidos
            for(int i = 0; i< 683; i++){
                proy_xy[i,0] = -1;
                proy_xy[i,1] = -1;
                proy_pol[i,0] = -1;
                proy_pol[i,1] = -1;
            }
            //se crean arreglos temporales para almacenar puntos
            float[,] temp_xy = new float[(mapa_polar.Length/2), 2];
            float[,] temp_pol = new float[(mapa_polar.Length/2), 2];

            for (int i = 0; i < (mapa_polar.Length/2); i++)
            {
                //verifica que valor de coordenadas anteriores no sea inválido
                if (mapa_polar[i,0]!=-1)
                {
                    temp_xy[i,0]= mapa_xy[i,0];
                    temp_xy[i,1]= mapa_xy[i,1];
                    temp_pol[i,0] = mapa_polar[i,0];
                    temp_pol[i,1]  = mapa_polar[i,1];
    //se verifica que puntos del mapa global, están dentro del rango angular específico 0 - 240.0
                    // y dentro del alcanze r<150 del sensor
                    if ((int)temp_pol[i, 0] >= 150 ||
                         Math.Ceiling((double)((temp_pol[i,1]*683.0)/240))>682)
                    {
                        temp_xy[i, 0] = -1;
                        temp_xy[i, 1] = -1;
                        temp_pol[i, 0] = -1;
                        temp_pol[i, 1] = -1; //banderas punto no existe o es no visible
                    }
                }
                else
                {
                    temp_xy[i, 0] = -1;
```

```
                    temp_xy[i, 1] = -1;
                    temp_pol[i, 0] = -1;
                    temp_pol[i, 1] = -1; //banderas punto no existe o es no visible
                }
            }

            //ahora se debe verificar que no haya superficies que no se pueden ver
            // (secuencias de angulos en sentido opuesto o puntos con mismo angulo (se tapan))
            int cuentaT;
            ArrayList js = new ArrayList();
            for (int i = 0; i < (mapa_polar.Length / 2); i++)
            {
                cuentaT = 0;
                js.Clear();
                for (int j = 0; j < (mapa_polar.Length / 2); j++)
                {
                    //se revisa si dos puntos tienen el mismo angulo
                    if ((int)((temp_pol[i, 1]*683.0)/240) == (int)((temp_pol[j, 1]*683.0)/240))
                    {
                        cuentaT++;
                        js.Add(j);
                    }
                }
                //si algun angulo esta repetido, se borra el punto mas lejano
                if (cuentaT > 1)
                {
                    int smallest=(int)js[0];
                    for(int k=1; k<js.Count; k++)
                    {
                     //se determina el punto más cercano al origen (distancia o radio mas pequeño)
                        if(temp_pol[(int)js[k],0]<temp_pol[smallest, 0])
                            smallest =(int)js[k];
                    }
                    //se borra los demás puntos con el mismo ángulo, solo se deja el más pequeño
                    for (int k = 0; k < js.Count; k++)
                    {
                        if ((int)js[k] != smallest)
                        {
                            temp_pol[(int)js[k], 0] = -1;
                            temp_pol[(int)js[k], 1] = -1;
                            temp_xy[(int)js[k], 0] = -1;
                            temp_xy[(int)js[k], 1] = -1;
                        }
                    }
                }
            }
            ArrayList invalid_index = new ArrayList();
            bool flag = false;
        //ahora se busca secuencias de ángulos que vayan en sentido contrario al establecido (ccw)
            for (int i = 0; i < (mapa_polar.Length / 2); i++)
            {
                if (i < ((mapa_polar.Length / 2) - 1))
                {
                    if (temp_pol[i, 1] != -1 && temp_pol[i + 1, 1] != -1)
                    {
                        do
                        {
                            if (temp_pol[i, 1] != -1 && temp_pol[i + 1, 1] != -1)
                            {
                                if ((i+1) == ((mapa_polar.Length / 2) - 1))
                                {
                                    //si ya se llego al último punto, se sale del ciclo
                                    flag = false;
                                }
                                else
                                {
                                    //cuando detecta un decremento en el ángulo
                                    if (temp_pol[i, 1] > temp_pol[i + 1, 1])
                                    {
```

```
                                    invalid_index.Add(i + 1); // primer punto inválido
                                    i++;
                                    flag = true;
                                }
                                else
                                {
                                    flag = false;
                                }
                            }
                        }
                        else
                        {
                            flag = false;
                        }
                    } while (flag);
                    //deja la i en el siguiente punto válido
                    if (invalid_index.Count != 0)
                    {
                        i = (int)invalid_index[invalid_index.Count - 1];
                        //se eliminan puntos inválidos
                        for (int j = 0; j < (invalid_index.Count - 1); j++)
                        {
                            temp_pol[(int)invalid_index[j], 0] = -1;
                            temp_pol[(int)invalid_index[j], 1] = -1;
                            temp_xy[(int)invalid_index[j], 0] = -1;
                            temp_xy[(int)invalid_index[j], 1] = -1;
                        }
                        //borra indices inválidos
                        invalid_index.Clear();
                    }
                }
            }
        }

        //ahora ya se tiene los puntos proyectados válidos
        //Se trasladan a arreglo de tamaño de escaneado
        for (int i = 0; i < (mapa_polar.Length / 2); i++)
        {
            if (temp_pol[i, 0] != -1)
            {
                int ind = (int)((mapa_polar[i, 1] * 683.0) / 240);
                if (ind >= 683)
                {
                    ind=682;
                }
                if (ind < 0)
                    ind = 0;
                proy_xy[ind,0] = temp_xy[i, 0];
                proy_xy[ind, 1] = temp_xy[i, 1];
                proy_pol[ind, 0] = temp_pol[i, 0];
                proy_pol[ind, 1] = temp_pol[i, 1];
                indexProy.Add(ind); //agrega indice en proyección
                indexGlob.Add(i); //agrega indice en mapa global
            }
        }
    }

    // ALGORITMO DE CAMBIO DE COORDENADAS
    // traslada centro o referencia de mapa global a nuevo punto
    // realiza proyección de escaneado anterior para alineación de nuevo escaneado
    private void traslate_map(float x_n, float y_n)
    {
        int maxData = mapa_polar.Length/2;
        for (int i = 0; i < maxData; i++)
        {
            //verifica que valor de coordenadas anteriores no sea invalido
            if(mapa_polar[i,0]!=-1)
            {
                //coordenadas cartesianas de los puntos proyectados en nueva posición
```

```
                    mapa_xy[i, 0] = x0 + mapa_xy[i, 0] - x_n;
                    mapa_xy[i, 1] = y_n - (y0 - mapa_xy[i, 1]); //en plano normal (y+ arriba)
                    //calcula coordenadas polares para cada punto - distancia , ángulo en grados
                    //distancia
                    mapa_polar[i, 0] = (float)distancia2(mapa_xy[i, 0],mapa_xy[i, 1], 0, 0);
                    //ángulo
                    mapa_polar[i, 1] = (float)(angle(0, 0, mapa_xy[i, 0], mapa_xy[i, 1]) *
                                        180 / Math.PI);
                }
                else
                {
                    mapa_xy[i, 0] = -1;
                    mapa_xy[i, 1] = -1;

                    mapa_polar[i, 0] = -1;
                    mapa_polar[i, 1] = -1; //banderas punto no existe o es no visible

                }
                //monitoreo2.Text += ("a: " + mapa_polar[i, 1] + "r: " + mapa_polar[i, 0] +"\n");
                //monitoreo3.Text += ("x: " + mapa_xy[i, 0] + "y: " + mapa_xy[i, 1] + "\n");
            }
        }

        //ALGORITMO DE UNIÓN Y ELIMINACIÓN DE REPETICIONES
        //Este método se aplica al terminar de ajustar a la nueva  posición,
        // se junta escaneado nuevo, con mapa transformado
        private void incrementMap()
        {
            //se traslada mapa global a arraylist para incrementar su tamaño
            ArrayList globalTempAngle = new ArrayList();
            ArrayList globalTempRad = new ArrayList();
            for (int i = 0; i < (mapa_polar.Length) / 2; i++)
            {
                bool elim = false;
                //no se añaden puntos proyectados
                for (int j = 0; j < indexGlob.Count; j++)
                {
                    if (i == (int)indexGlob[j])
                    {
                        elim = true;
                    }
                }
                if (!elim)
                {
                    globalTempAngle.Add(mapa_polar[i, 1]);
                    globalTempRad.Add(mapa_polar[i, 0]);
                }
            }
            indexProy.Clear();
            indexGlob.Clear();

            //ahora se añaden puntos de escaneado nuevo
            for (int i = 0; i < dist.Length; i++)
            {
                if (dist[i] != -1)
                {
                    float angul = (float)((i * 240.0) / 683);
                    globalTempAngle.Add(angul);
                    globalTempRad.Add(dist[i]);
                }
            }
            //ahora se regresa a mapa polar todos los datos, y se recalcula mapa cartesiano
            mapa_polar = new float[globalTempAngle.Count, 2];
            mapa_xy = new float[globalTempAngle.Count, 2];
            for (int i = 0; i < globalTempAngle.Count; i++)
            {
                mapa_polar[i, 0] = (float)globalTempRad[i];
                mapa_polar[i, 1] = (float)globalTempAngle[i];
                //calculo de mapa cartesiano
```

```
                mapa_xy[i, 0] = (float)(mapa_polar[i, 0] * Math.Cos((mapa_polar[i, 1] * Math.PI)
                                / 180));
                mapa_xy[i, 1] = (float)(mapa_polar[i, 0] * Math.Sin((mapa_polar[i, 1] * Math.PI)
                                /180));
            }
        }

        //algoritmo de alignment and matching para formar mapa global
        private void point_correspondence_matching()
        {
            //closest point rule
            int corresp = 0;
            double dist_min = 1000.0;
            double dista = 0.0;
            double[] dist_par = new double[683];
            //se identifica punto en proyeccion con distancia mínima
            // y se asigna como pareja a punto de escaneado nuevo
            for (int i = 0; i < 683; i++)
            {
                corresp = 0;
                dist_min = 1000.0;
                dista = 0.0;

                if (dist[i] != -1)
                {
                    for (int j = 0; j < 683; j++)
                    {
                        if (proy_pol[j, 0] != -1 && proy_pol[j, 1] != -1)
                        {

                                dista = distancia2(dist_xy[i, 0], dist_xy[i, 1], proy_xy[j, 0],
                                proy_xy[j, 1]);
                            if (dista <= dist_min)
                            {
                                corresp = j;
                                dist_min = dista;
                            }
                        }
                    }
                    //cuando se encontró el punto con distancia mínima, se guarda su índice en P1
                    P1[i] = corresp;
                    dist_par[i] = dist_min;

                    //Se elimina puntos fuera de rango angular Bw
                    if (Math.Abs((i * 240.0) / 683 - proy_pol[P1[i], 1]) > Bw)
                    {
                        P1[i] = -1;
                        dist_par[i] = -1;
                        for (int a=0; a<indexProy.Count; a++){
                            if((int)indexProy[a]==i){
                                indexProy.RemoveAt(a);
                                indexGlob.RemoveAt(a); //elimina de arreglo de omisión
                            }
                        }

                    }

                }
                else
                {
                    P1[i] = -1;
                    dist_par[i] = -1;
                }
            }

            //Ahora se eliminan puntos que tienen al mismo correspondiente asignado,
            // la relación tiene que ser 1 a 1
            for (int k = 0; k < 683; k++)
            {
```

```
                if(P1[k]!=-1){
                    for (int l = 0; l < 683; l++)
                    {
                        if (P1[k] == P1[l] && k!=l) //si tienen misma pareja (y no es mismo punto)
                        {
                            if (dist_par[k] < dist_par[l]) //se deja solo el de distancia menor
                            {
                                dist_par[l] = -1;
                                P1[l] = -1;
                                //borra punto de arreglo de omisión
                                for (int a = 0; a < indexProy.Count; a++)
                                {
                                    if ((int)indexProy[a] ==l)
                                    {
                                        indexProy.RemoveAt(a);
                                        indexGlob.RemoveAt(a); //elimina de arreglo de omisión
                                    }
                                }
                            }
                        }
                    }
                }
            }

            //ahora se guardan distancias de puntos válidos
            for (int i = 0; i < 683; i++)
            {
                if (dist_par[i] != -1)//si es valido, agrega distancia a arreglo
                    dist_m.Add(dist_par[i]);
            }

            //Se eliminan los puntos que esten mas alla de un treshold Br
            Br = 150 / 10;                                              //r/10
            //y se procede a eliminar los datos mas lejanos al límite
            for (int l = 0; l < P1.Length; l++)
            {
                //si el punto tiene pareja
                if (P1[l] != -1)
                {
                    //y si la distancia entre los puntos es mayor al threshold o a 1/3 del rango d
                    if (distancia2(dist_xy[l, 0], dist_xy[l, 1], proy_xy[P1[l], 0],
                          proy_xy[P1[l], 1]) > Br || distancia2(dist_xy[l, 0], dist_xy[l, 1],
                    proy_xy[P1[l], 0], proy_xy[P1[l], 1])>150/3){
                        P1[l] = -1; //se borra el punto
                        //y se elimina punto de arreglo de omisión
                        for (int a = 0; a < indexProy.Count; a++)
                        {
                            if ((int)indexProy[a] == l)
                            {
                                indexProy.RemoveAt(a);
                                indexGlob.RemoveAt(a); //elimina de arreglo de omisión
                            }
                        }
                    }
                }
            }
            //ahora ya se tiene el conjunto de puntos correspondientes por
            //la regla de punto mas cercano modificada
            //P1 guarda indices de puntos correspondientes en Sref. P1[a] = b.
            // a es el indice de Snew, b es el indice de Sref

            //Se procede a calcular la solución de mínimos cuadrados (w1, T1)
            // a partir de los pares correspondientes con P1

            closed_Form_PbM1();

            //mantiene record de errores de cada iteracion
            monitoreo4.Text += "\n w residual(w2): " + w1 * 180 / Math.PI;
            monitoreo4.Text += "\n T residual(Tx): " + Tx1 + " (Ty): " + Ty1;
```

```
        }

        //Forma cerrada de la solución de mínimos cuadrados , da valores a Tx, Ty y w
        private void closed_Form_PbM1()
        {
            double x_m = 0.0, y_m = 0.0, x_m1 = 0.0, y_m1 = 0.0;
            double Sxx1 = 0.0, Syy1 = 0.0, Sxy1 = 0.0, Syx1 = 0.0;

            int cuenta1 = 0;
            //se calculan primeros parámetros
            for (int a = 0; a < 683; a++)
            {
                if (dist[a] != -1 && P1[a] != -1)
                {
                    x_m += dist_xy[a,0];
                    y_m += dist_xy[a,1];
                    x_m1 += proy_xy[P1[a], 0];
                    y_m1 += proy_xy[P1[a], 1];
                    cuenta1++;
                }
            }
            x_m = x_m / cuenta1;
            y_m = y_m / cuenta1;
            x_m1 = x_m1 / cuenta1;
            y_m1 = y_m1 / cuenta1;

            //se calcula segundo set de parámetros
            for (int a = 0; a < 683; a++)
            {
                if (dist[a] != -1 && P1[a] != -1)
                {
                    Sxx1 += (dist_xy[a,0] - x_m) * (proy_xy[P1[a], 0] - x_m1);
                    Syy1 += (dist_xy[a,1] - y_m) * (proy_xy[P1[a], 1] - y_m1);
                    Sxy1 += (dist_xy[a,0] - x_m) * (proy_xy[P1[a], 1] - y_m1);
                    Syx1 += (dist_xy[a,1] - y_m) * (proy_xy[P1[a], 0] - x_m1);
                }
            }

            //con todos los parámetros necesarios, se encuentra rotación y traslación
            w1 = Math.Atan(((Sxy1 - Syx1)/ (Sxx1 + Syy1)));
            Tx1 = x_m1 - (x_m * Math.Cos(w1) - y_m * Math.Sin(w1));
            Ty1 = y_m1 - (x_m * Math.Sin(w1) + y_m * Math.Cos(w1));
        }

        //ALGORITMO DE TRANSFORMACIÓN (aplica Tx, Ty y w a mapa global)
        //metodo para aplicar transformacion a mapa global, y poder realizar
        // iteración de método de alineación
        private void apply_transform(double ang, double TrX, double TrY)
        {
            double angDegree = ((ang * 180.0) / Math.PI); //transformacion angular en grados
            //se crean arreglos temporales para almacenar puntos transformados
            float[,] temp_xy = new float[(mapa_polar.Length / 2), 2];
            float[,] temp_pol = new float[(mapa_polar.Length / 2), 2];

            //se aplica transformación polar
            for (int i = 0; i < (mapa_polar.Length / 2); i++)
            {
                if (mapa_polar[i, 0] != -1 && mapa_polar[i, 1] != -1)
                {
                    temp_pol[i, 0] = mapa_polar[i, 0];
                    temp_pol[i, 1] = (float)(mapa_polar[i, 1] - angDegree);
                    if (temp_pol[i, 1] < 0)
                    {
                        temp_pol[i, 1] = 360 + temp_pol[i, 1];
                    }
                    if (temp_pol[i, 1] >= 360)
                    {
                        temp_pol[i, 1] = (temp_pol[i, 1] - 360);
                    }
```

```
                    temp_xy[i, 0] = (float)(temp_pol[i, 0] * Math.Cos((temp_pol[i, 1]
                                    *Math.PI)/180));
                    temp_xy[i, 1] = (float)(temp_pol[i, 0] * Math.Sin((temp_pol[i, 1] * Math.PI)
                                    / 180));
                }
                else
                {
                    temp_pol[i, 0] = -1;
                    temp_pol[i, 1] = -1;
                    temp_xy[i, 0] = -1;
                    temp_xy[i, 1] = -1;
                }
            }

            //Ahora se aplica transformación horizontal TrX y vertical TrY
            for (int i = 0; i < (mapa_polar.Length / 2); i++)
            {
                if (mapa_polar[i, 0] != -1 && mapa_polar[i, 1] != -1)
                {
                    temp_xy[i, 0] = (float)(temp_xy[i, 0] - TrX);
                    temp_xy[i, 1] = (float)(temp_xy[i, 1] - TrY);
                    temp_pol[i, 0] = (float)distancia2(temp_xy[i, 0], temp_xy[i, 1], 0, 0);
                                                                        //distancia
                    temp_pol[i, 1] = (float)(angle(0, 0, temp_xy[i, 0], temp_xy[i, 1]) *
                                    180 / Math.PI);
                }
                else
                {
                    temp_pol[i, 0] = -1;
                    temp_pol[i, 1] = -1;
                    temp_xy[i, 0] = -1;
                    temp_xy[i, 1] = -1;
                }
            }

            //se guardan valores en mapa global
            for (int i = 0; i < (mapa_polar.Length / 2); i++)
            {
                mapa_polar[i,0] = temp_pol[i, 0];
                mapa_polar[i,1] = temp_pol[i, 1];
                mapa_xy[i, 0] = temp_xy[i, 0];
                mapa_xy[i, 1] = temp_xy[i, 1];

            }
        }

        //Estrategia de exploración basada en información
        //método para calcular siguiente posición con algoritmo de Amigoni Caglioti
        private void algoritmo_AC(int angr)
        {
            //se generan puntos candidatos
            //tecnica A, datos en circulo r/2, separados de manera pareja (cada 15º) 17 puntos
            ArrayList candX = new ArrayList();
            ArrayList candY = new ArrayList();
            for (int i = 0; i < 17; i++)
            {
                float angul = (float)(i * (683.0 / 16) * (240.0 / 683));
                float canX = (float)((r/2) * Math.Cos((angul* Math.PI) / 180)); //x
                float canY = (float)((r / 2) * Math.Sin((angul * Math.PI) / 180)); //y
                //y se eliminan candidatos invalidos(pasan a traves de paredes u objetos)
                float angu_plot = (float)((i * (683.0 / 16) * (240.0 / 683)) + angr);
                float canX_plot =(float)((r/2) * Math.Cos((angu_plot* Math.PI) / 180));
                float canY_plot =(float)((r / 2) * Math.Sin((angu_plot * Math.PI) / 180));
                plotCandidatos(canX_plot, canY_plot);
                if(check_path(canX_plot, canY_plot)){
                    candX.Add(canX);
                    candY.Add(canY);
                }
            }
```

```
            //ahora se almacenan datos en arreglo
            candidatos = new float[candX.Count, 2];

            for (int i = 0; i < candX.Count; i++)
            {
                candidatos[i, 0] = (float)candX[i];
                candidatos[i, 1] = (float)candY[i];
                monitoreo2.Text += "X: " + candidatos[i, 0] + " - Y: " + candidatos[i, 1] + "\n";
            }

            float coorXOrigi = x0;
            float coorYOrigi = y0;
            double[] J = new double[candidatos.Length / 2];
            monitoreo3.Text = "";
            //se aplica criterio a candidatos válidos
            for (int a = 0; a < candidatos.Length / 2; a++)
            {
                //Se traslada mapa global a posición candidata
                traslate_map(x0 + candidatos[a, 0], y0-candidatos[a, 1]);
                //Se realiza proyección
                proyeccion();

                float[,] A = new float[proy_xy.Length / 2, 2];
                int AA=0;
                for (int i = 0; i < proy_xy.Length / 2; i++)
                {
                    A[i, 0] = proy_xy[i, 0];
                    A[i, 1] = proy_xy[i, 1];
                    if (proy_xy[i, 0] != -1 && proy_xy[i, 1] != -1)
                        AA++;
                }
                int N = 683 - AA; //estimación optimista de puntos nuevos a encontrar en posición
                double V0 = 0.01;
                double Vxy = 0.001;
                double sigma_unci = V0 * Math.Pow((r / 2), 2) + Vxy;
                double sigma = 0.0001;
                double sigma_p = 2 * sigma;
                double c = r / 2;
                double sum_sigma_unci = 0.0;
                double P = 3000;
                for (int i = 0; i < 683; i++)
                {
                    sum_sigma_unci += Math.Log(sigma_unci / sigma);
                }
                double sum_sigma_p = 0.0;
                for (int i = 0; i < AA / 2; i++)
                {
                    sum_sigma_p += Math.Log(sigma / sigma_p);
                }
                J[a] = (1.0 / (N + A.Length / 2)) * sum_sigma_unci + N * Math.Log(sigma / P) +
                    sum_sigma_p + N * Math.Log((2 * Math.PI) / sigma);
                traslate_map(x0 - (candidatos[a, 0]), y0 + candidatos[a, 1]);
                monitoreo3.Text += "valor J: " + J[a] + "\n";
            }
            double minJ = 10000.0;
            for (int i = 0; i < candidatos.Length / 2; i++)
            {
                //Se selecciona como siguiente posición el punto con menor criterio J
                if (J[i] < minJ)
                {
                    minJ = J[i];
                    nextPx = candidatos[i, 0];
                    nextPy = candidatos[i, 1];
                }
            }
        }

        //MÉTODOS PARA GRAFICAR
        /*-------------------------------------------------------------------------------------*/
```

```
        //se dibuja mapa local con datos de escaneado
        int xl0 = 0, yl0 = 0;
        private void localMap()
        {
            xl0 = myBitmap2.Size.Width / 2;
            yl0 = myBitmap2.Size.Height / 2;
            double xl_ = 0.0, yl_ = 0.0;
            double angulo;
            for (int i = 0; i < 683; i++)
            {
                angulo = (i*(240.0/683))* Math.PI / 180;
                xl_ = xl0 + dist_xy[i,0];
                yl_ = yl0 - dist_xy[i,1]; //eje y esta invertido

                if (dist[i] != -1){
                    myBitmap2.SetPixel((int)xl_, (int)yl_, Color.Blue);
                }
            }

            //dibuja posicion del robot
            Graphics graphicsObj;
            graphicsObj = Graphics.FromImage(myBitmap2);
            graphicsObj.DrawEllipse(pen, xl0 - 2, yl0 - 2, 4, 4);

            graphicsObj.DrawImage(myBitmap2, 0, 0, myBitmap2.Width, myBitmap2.Height);
            graphicsObj.Dispose();
            pictureBox2.Image = (Image)myBitmap2;

        }

        //se dibuja proyección de escaneado anterior en la siguiente posición
        private void plotProy()
        {
            xl0 = myBitmap2.Size.Width / 2;
            yl0 = myBitmap2.Size.Height / 2;

            double xl_ = 0.0, yl_ = 0.0;
            double angulo;

            for (int i = 0; i < 683; i++)
            {

                if (proy_pol[i,0]!=-1 && proy_pol[i,1]!=-1)
                {
                    angulo = proy_pol[i,1] * Math.PI / 180.0;
                    xl_ = xl0 + proy_xy[i,0];
                    yl_ = yl0 - proy_xy[i,1]; //eje y invertido

                    myBitmap2.SetPixel((int)xl_, (int)yl_, Color.Red);
                }
            }
            pictureBox2.Image = (Image)myBitmap2;

        }

        //mapa para plotear mapa global actual
        private void plotGlobal()
        {
            xl0 = myBitmap2.Size.Width / 2;
            yl0 = myBitmap2.Size.Height / 2;
            double xl_ = 0.0, yl_ = 0.0;
            int maxData = mapa_polar.Length/2;
            for (int i = 0; i < maxData; i++)
            {
                xl_ = xl0 + mapa_xy[i, 0];
                yl_ = yl0 - mapa_xy[i, 1]; //eje y esta invertido

                if (mapa_polar[i, 0] != -1 && (xl_ > 0 && xl_ < myBitmap2.Size.Width)
                    && (yl_ > 0 && yl_ < myBitmap2.Size.Height))
```

```
                {
                    myBitmap2.SetPixel((int)xl_, (int)yl_, Color.ForestGreen);
                }
            }

            //dibuja posicion del robot
            Graphics graphicsObj;
            graphicsObj = Graphics.FromImage(myBitmap2);
            graphicsObj.DrawEllipse(pen, xl0 - 2, yl0 - 2, 4, 4);

            graphicsObj.DrawImage(myBitmap2, 0, 0, myBitmap2.Width, myBitmap2.Height);
            graphicsObj.Dispose();
            pictureBox2.Image = (Image)myBitmap2;

        }

        private void plotCandidatos(float xx, float yy)
        {
                float xl_=0, yl_=0;
                xl_ = x0 + xx;
                yl_ = y0 - yy; //eje y esta invertido
                try
                {
                    myBitmap.SetPixel((int)xl_, (int)yl_, Color.Red);
                }
                catch (Exception e)
                {}

            //dibuja posicion del robot
            Graphics graphicsObj;
            graphicsObj = Graphics.FromImage(myBitmap);
            graphicsObj.DrawImage(myBitmap, 0, 0, myBitmap.Width, myBitmap.Height);
            graphicsObj.Dispose();
            pictureBox1.Image = (Image)myBitmap;
        }

    }
}
```

4. CÓDIGO FINAL – ALGORITMOS APLICADOS A ROBOHELICÓPTERO

Lenguaje de programación: C#

```
/*
 *  Autor: Roberto Enrique Saravia Fernández
 *  Archivo: mapeo.cs
 *  Descripción: métodos con algoritmos utilizados para mapeo y exploración
 */
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections;

namespace WindowsFormsApplication1
{
    class Mapeo
    {
        // mapa global creado
            // coord. polares [a,0] r - [a,1] ángulo en grados
        public float[,] mapa_polar = new float[228,2];
            // coord. cartesianas relativas a posición actual
            // [a,0] x - [a,1] y
        public float[,] mapa_xy = new float[228, 2];

        //puntos candidatos para siguiente posición
            // coordenadas x y y con respecto al punto actual
        public float[,] candidatos;

        //datos para traslación temporal
            // coord. polares [a,0] r - [a,1] ángulo en grados
        public float[,] trasl_polar;
            // coord. cartesianas relativas a posición actual
        public float[,] trasl_xy;

        //datos para proyección
            //coord. cartesianas de cada punto proyectado
            // respecto a nueva posición
        float[,] proy_xy = new float[228, 2];

        int[] dist_n = new int[228];
            // coord. polares de cada punto proyectado
        float[,] proy_pol = new float[228, 2];
            // índice en proyección de cada punto proyectado
        ArrayList indexProy = new ArrayList();
            // índice en mapa global de cada punto proyectado
        ArrayList indexGlob = new ArrayList();

        //datos de mapa anterior local
            // coord. polares de mapa local
        public float[,] ante_polar = new float[228, 2];
            // coord. cartesianas relativas a posición actual
        public float[,] ante_xy = new float[228, 2];

        //datos de mapa local de nueva posición
            //coordenadas polares de mapa local
        public float[,] local_polar = new float[228, 2];
            //coordenadas cartesianas relativas a posición actual
        public float[,] local_xy = new float[228, 2];
```

```
        //datos de algoritmo alignment and matching
            //rango máximo que delimita sector de búsqueda
        private int Bw = 10;
            // arreglo de puntos correspondientes por distancia mínima
            // (almacena índices)
        int[] P1 = new int[228];
            // rotación obtenida de regla de punto más cercano
        double w1 = 0.0;
            //traslación obtenida de regla del punto más cercano
        double Tx1 = 0.0, Ty1 = 0.0;

        //CONSTRUCTOR, recibe primer escaneo de robot
        public Mapeo(int[] arr){
            //se almacena primer segmento de mapa,
            // correspondiente a primera escaneada
            for (int i = 0; i < 228; i++)
            {
                if (arr[i] < 4095)
                {
                    mapa_polar[i, 0] = arr[i]; //r
                    mapa_polar[i, 1] = (float)((i * 240.0) / 228); //ángulo
                    mapa_xy[i, 0] = (float)(arr[i] *
                        Math.Cos(mapa_polar[i, 1] * Math.PI / 180)); //x
                    mapa_xy[i, 1] = (float)(arr[i] *
                        Math.Sin(mapa_polar[i, 1] * Math.PI / 180)); //y

                    //primer mapa local
                    local_polar[i, 0] = arr[i]; //r
                    local_polar[i, 1] = (float)((i * 240.0) / 228); //ángulo
                    local_xy[i, 0] = (float)(arr[i] *
                        Math.Cos(local_polar[i, 1] * Math.PI / 180)); //x
                    local_xy[i, 1] = (float)(arr[i] *
                        Math.Sin(local_polar[i, 1] * Math.PI / 180)); //y
                }
            }
        }

        //método para recibir mapa local en nueva posición,
        // y guardar mapa anterior local
        public void localMap(int[] arr)
        {
            for (int i = 0; i < 228; i++)
            {
                if (arr[i] < 4095)
                {
                    //guarda local anterior
                    ante_polar[i, 0] = local_polar[i, 0]; //r
                    ante_polar[i, 1] = local_polar[i, 1]; //ángulo
                    ante_xy[i, 0] = local_xy[i, 0]; //x
                    ante_xy[i, 1] = local_xy[i, 1]; //y

                    //y guarda mapa local nuevo
                    local_polar[i, 0] = arr[i]; //r
                    local_polar[i, 1] = (float)((i * 240.0) / 228); //ángulo
                    local_xy[i, 0] = (float)(arr[i] *
                        Math.Cos(local_polar[i, 1] * Math.PI / 180)); //x
                    local_xy[i, 1] = (float)(arr[i] *
                        Math.Sin(local_polar[i, 1] * Math.PI / 180)); //y
                }
            }
        }


        //método para calcular mejor siguiente posición
        // en base a estrategia de exploración elegida
        public float[] sigPos(int tec){
            //Se aplica la técnica seleccionada
            float[] np = new float[2];
            if (tec == 0) //random
```

```
            {
                //se generan valores teóricos de nueva posición
                // x1 = e.X;
                // y1 = e.Y;
                /* x1 = ran1.Next(0, 100) * pictureBox1.Size.Width / 100;
                 y1 = ran1.Next(0, 100) * pictureBox1.Size.Height / 100;*/
            }
            if (tec == 1)
            { // algoritmo de exploración basado en información
                // calcula siguiente mejor posición en base a mapa actual
                np = algoritmo_AC();
            }
            return np;
        }

        //traslada e incrementa mapa global en nueva posición
        // (con dato de odometría)
        public float[] move(int x, int y)
        {
            float[] tr = new float[3];
            tr[0] = 0;  //Tx
            tr[1] = 0;  //Ty
            tr[2] = 0;  // rotacion

            //se asume que no se mueve con respecto a escaneado anterior
            //con mapa local anterior, se realiza alineación de nuevo escaneado
            proyeccion(ante_xy, ante_polar);

            //se realiza alineación de nuevo escaneado con respecto a la proyección
            // (dato odometría)
            //y se encuentra la traslación y rotación real del movimiento.
            int count_a = 15;   //se redujo a 15 para mejorar tiempo de ejecución
            Bw = 10;
            do
            {
                point_correspondence_matching();
                if (double.IsNaN(w1) || double.IsNaN(Tx1))
                {
                    tr[0] += 0;
                    tr[1] += 0;
                    tr[2] += 0;
                }
                else
                {
                    tr[0] += (float)Tx1;
                    tr[1] += (float)Ty1;
                    tr[2] += (float)w1;
                  //  apply_transform(w1, Tx1, Ty1);
                  // se aplica transformación w y Tx, Ty a mapa global si existe
                    apply_transform_ante(w1, Tx1, Ty1);
                }
                proyeccion(ante_xy, ante_polar);
             //   proyeccion(mapa_xy, mapa_polar);
                //plotGlobal();
                //globalMap();
                count_a--;
            } while (count_a != 0);

            // se aplica transformación w y Tx, Ty a mapa global si existe
            apply_transform(tr[2], tr[0], tr[1]);
            // se genera proyección del mapa
            proyeccion(mapa_xy, mapa_polar);
            // y se incrementa el mapa global
            increment_GM();
            //Se tiene mapa nuevo

            float radial = (float)Math.Sqrt(Math.Pow(tr[0], 2)
                           + Math.Pow(tr[1], 2));
            float angul = (float)Math.Atan2(tr[1] , tr[0]);
```

```
            tr[0] = (float)(radial * Math.Cos(angul - 30*Math.PI/180.0));
            tr[1] = (float)(radial * Math.Sin(angul - 30*Math.PI/180.0));

            return tr;
        }

        //ALGORITMO DE UNIÓN Y ELIMINACIÓN DE REPETIDOS
        //metodo para agrandar mapa global,
        // se manda como parámetro el mapa local nuevo
        void increment_GM()
        {
            //se traslada mapa global a arraylist para incrementar su tamaño
            ArrayList globalTempAngle = new ArrayList();
            ArrayList globalTempRad = new ArrayList();
            for (int i = 0; i < (mapa_polar.Length) / 2; i++)
            {
                bool elim = false;
                //no se añaden puntos proyectados
                for (int j = 0; j < indexGlob.Count; j++)
                {
                    if (i == (int)indexGlob[j])
                    {
                        elim = true;
                    }
                }
                if (!elim)
                {
                    globalTempAngle.Add(mapa_polar[i, 1]);
                    globalTempRad.Add(mapa_polar[i, 0]);
                }
            }
            indexProy.Clear();
            indexGlob.Clear();

            //ahora se añaden puntos de escaneado nuevo
            for (int i = 0; i < local_polar.Length/2; i++)
            {
                if (local_polar[i,0] != -1)
                {
                    float angul = (float)((i * 240.0) / 228);
                    globalTempAngle.Add(angul);
                    globalTempRad.Add(local_polar[i,0]);
                }
            }
            //ahora se regresa a mapa polar todos los datos,
            // y se recalcula mapa cartesiano
            mapa_polar = new float[globalTempAngle.Count, 2];
            mapa_xy = new float[globalTempAngle.Count, 2];
            for (int i = 0; i < globalTempAngle.Count; i++)
            {
                mapa_polar[i, 0] = (float)globalTempRad[i];
                mapa_polar[i, 1] = (float)globalTempAngle[i];
                //calculo de mapa cartesiano
                mapa_xy[i, 0] = (float)(mapa_polar[i, 0] *
                        Math.Cos((mapa_polar[i, 1] * Math.PI) / 180));
                mapa_xy[i, 1] = (float)(mapa_polar[i, 0] *
                        Math.Sin((mapa_polar[i, 1] * Math.PI) / 180));
            }
        }

        // ESTRATEGIA DE EXPLORACIÓN BASADA EN INFORMACIÓN
        //algoritmo que calcula mejor posición
        private float[] algoritmo_AC()
        {
            //se generan puntos candidatos
            //tecnica A, datos en circulo r/2, separados de manera pareja
            // (cada 15 grados) son 17 puntos
            ArrayList candX = new ArrayList();
```

```
            ArrayList candY = new ArrayList();
            int r = 4095;
            for (int i = 0; i < 17; i++)
            {
                float angul = (float)(i * (228.0 / 16) * (240.0 / 228));
                // movimientos de 40 centímetros
                float canX = (float)((r / 10) *
                                Math.Cos((angul * Math.PI) / 180)); //x
                float canY = (float)((r / 10) *
                                Math.Sin((angul * Math.PI) / 180)); //y
                //y se eliminan candidatos inválidos
                // (pasan a traves de paredes u objetos)
                if (check_path(canX, canY))
                {
                    candX.Add(canX);
                    candY.Add(canY);
                }
            }
            //ahora se almacenan datos en arreglo
            candidatos = new float[candX.Count, 2];

            for (int i = 0; i < candX.Count; i++)
            {
                candidatos[i, 0] = (float)candX[i];
                candidatos[i, 1] = (float)candY[i];
            }

            double[] J = new double[candidatos.Length / 2];
            //se aplica criterio a candidatos válidos
            for (int a = 0; a < candidatos.Length / 2; a++)
            {
                //Se traslada mapa global a posición candidata
                traslate_map(candidatos[a, 0], candidatos[a, 1]);
                //Se realiza proyección
                proyeccion(trasl_xy, trasl_polar);

                //contiene puntos existentes en candidato
                float[,] A = new float[proy_xy.Length / 2, 2];
                int AA = 0;
                for (int i = 0; i < proy_xy.Length / 2; i++)
                {
                    A[i, 0] = proy_xy[i, 0];
                    A[i, 1] = proy_xy[i, 1];
                    if (proy_xy[i, 0] != -1 && proy_xy[i, 1] != -1)
                        AA++;
                }
                int N = 228 - AA; //estimación optimista de puntos nuevos
                                  // a encontrar en nueva posición
                double V0 = 0.01;
                double Vxy = 0.001;
                double sigma_unci = V0 * Math.Pow((r / 2), 2) + Vxy;
                double sigma = 0.0001;
                double sigma_p = 2 * sigma;
                double c = r / 2;
                double sum_sigma_unci = 0.0;
                double P = 100000;
                for (int i = 0; i < 228; i++)
                {
                    sum_sigma_unci += Math.Log(sigma_unci / sigma);
                }
                double sum_sigma_p = 0.0;
                for (int i = 0; i < AA / 2; i++)
                {
                    sum_sigma_p += Math.Log(sigma / sigma_p);
                }
                //aplicación de criterio J
                J[a] = (1.0 / (N + A.Length / 2)) * sum_sigma_unci +
                        N * Math.Log(sigma / P) + sum_sigma_p +
                        N * Math.Log((2 * Math.PI) / sigma);
```

```
            }
            double minJ = 10000.0;
            float nextPx = 0;
            float nextPy = 0;
            //se elije al candidato con menor valor de J
            for (int i = 0; i < candidatos.Length / 2; i++)
            {
                if (J[i] < minJ)
                {
                    minJ = J[i];
                    nextPx = candidatos[i, 0];
                    nextPy = candidatos[i, 1];
                }
            }
            float[] p = new float[2];
            p[0] = nextPx;
            p[1] = nextPy;
            return p;
        }

        //ALGORITMO DE CAMBIO DE COORDENADAS
        //traslada centro o referencia de mapa global a nuevo punto
        //realiza proyección de escaneado anterior
        // para alineación de nuevo escaneado
        private void traslate_map(float x_n, float y_n)
        {
            int maxData = mapa_polar.Length / 2;
            trasl_xy = new float[maxData, 2];
            trasl_polar = new float[maxData, 2];

            for (int i = 0; i < maxData; i++)
            {
                //verifica que valor de coordenadas anteriores no sea inválido
                if (mapa_polar[i, 0] != -1)
                {
                    //se obtiene coordenadas cartesianas de los puntos
                    // proyectados en nueva posición
                    trasl_xy[i, 0] = mapa_xy[i, 0] - x_n;
                    trasl_xy[i, 1] = mapa_xy[i, 1] -  y_n;
                    //se calcula coordenadas polares para cada punto
                    //distancia
                    trasl_polar[i, 0] =
                        (float)distancia2(mapa_xy[i, 0], mapa_xy[i, 1], 0, 0);
                    //ángulo en grados
                    trasl_polar[i, 1] =
                        (float)(angle(0, 0, mapa_xy[i, 0], mapa_xy[i, 1]) * 180 / Math.PI);
                }
                else
                {
                    //banderas para indicar que este punto no existe o es no visible
                    trasl_xy[i, 0] = -1;
                    trasl_xy[i, 1] = -1;

                    trasl_polar[i, 0] = -1;
                    trasl_polar[i, 1] = -1;

                }
            }
        }
        //ALGORITMO DE PROYECCIÓN
        //se realiza proyección de mapa enviado como parámetro
        private void proyeccion(float[,] xy, float[,] polar)
        {
            //Se inicializan arreglos de proyección con valores de puntos inválidos
            for (int i = 0; i < 228; i++)
            {
                proy_xy[i, 0] = -1;
                proy_xy[i, 1] = -1;
                proy_pol[i, 0] = -1;
```

```
                proy_pol[i, 1] = -1;
            }
            //se crean arreglos temporales para almacenar puntos
            float[,] temp_xy = new float[(polar.Length / 2), 2];
            float[,] temp_pol = new float[(polar.Length / 2), 2];

            for (int i = 0; i < (polar.Length / 2); i++)
            {
                //verifica que valor de coordenadas anteriores no sea inválido
                if (polar[i, 0] != -1)
                {
                    //se verifica que puntos del mapa global, estan dentro
                    // del rango angular específico 0 - 240.0 y dentro
                    //del alcanze r<4096 mm
                    temp_xy[i, 0] = xy[i, 0];
                    temp_xy[i, 1] = xy[i, 1];
                    temp_pol[i, 0] = polar[i, 0];
                    temp_pol[i, 1] = polar[i, 1];
                    //se verifica que puntos no esten fuera del alcance del sensor,
                    // que es 4096 mm y rango de 240 grados
                    if ((int)temp_pol[i, 0] >= 4096 ||
                        Math.Ceiling((double)((temp_pol[i, 1] * 228.0) / 240)) > 227)
                    {
                        //banderas para indicar que este punto no existe o es no visible
                        temp_xy[i, 0] = -1;
                        temp_xy[i, 1] = -1;
                        temp_pol[i, 0] = -1;
                        temp_pol[i, 1] = -1;
                    }
                }
                else
                {
                    //banderas para indicar que este punto no existe o es no visible
                    temp_xy[i, 0] = -1;
                    temp_xy[i, 1] = -1;
                    temp_pol[i, 0] = -1;
                    temp_pol[i, 1] = -1;
                }
            }

            //ahora se debe verificar que no haya superficies que no se pueden ver
            // (secuencias de ángulos en sentido opuesto o puntos con mismo ángulo)
            int cuentaT;
            ArrayList js = new ArrayList();
            for (int i = 0; i < (polar.Length / 2); i++)
            {
                cuentaT = 0;
                js.Clear();
                for (int j = 0; j < (polar.Length / 2); j++)
                {
                    //se revisa si dos puntos tienen el mismo ángulo
                    if ((int)((temp_pol[i, 1] * 228.0) / 240) ==
                            (int)((temp_pol[j, 1] * 228.0) / 240))
                    {
                        cuentaT++;
                        js.Add(j);
                    }
                }
                //si algun ángulo esta repetido, se borra el punto más lejano
                if (cuentaT > 1)
                {
                    int smallest = (int)js[0];
                    for (int k = 1; k < js.Count; k++)
                    {
                        //se determina el punto mas cercano al origen
                        // (distancia o radio mas pequeño)
                        if (temp_pol[(int)js[k], 0] < temp_pol[smallest, 0])
                            smallest = (int)js[k];
                    }
```

```
                    //se borra los demás puntos con el mismo ángulo,
                    // solo se deja el más pequeño
                    for (int k = 0; k < js.Count; k++)
                    {
                        if ((int)js[k] != smallest)
                        {
                            temp_pol[(int)js[k], 0] = -1;
                            temp_pol[(int)js[k], 1] = -1;
                            temp_xy[(int)js[k], 0] = -1;
                            temp_xy[(int)js[k], 1] = -1;
                        }
                    }

                }
            }
            ArrayList invalid_index = new ArrayList();
            bool flag = false;
            //ahora se busca secuencias de ángulos que vayan
            // en sentido contrario al establecido (ccw)
            for (int i = 0; i < (polar.Length / 2); i++)
            {
                //Si no es el último punto
                if (i < ((polar.Length / 2) - 1))
                {
                    if (temp_pol[i, 1] != -1 && temp_pol[i + 1, 1] != -1)
                    {
                        do
                        {
                            if (temp_pol[i, 1] != -1 && temp_pol[i + 1, 1] != -1)
                            {
                                if ((i + 1) == ((polar.Length / 2) - 1))
                                {
                                    //si ya se llego al ultimo punto, se sale del ciclo
                                    flag = false;
                                }
                                else
                                {
                                    //cuando detecta un decremento en el ángulo
                                    if (temp_pol[i, 1] > temp_pol[i + 1, 1])
                                    {
                                        //se agrega primer punto inválido
                                        invalid_index.Add(i + 1);
                                        i++;
                                        flag = true;
                                    }
                                    else
                                    {
                                        flag = false;
                                    }
                                }
                            }
                            else
                            {
                                flag = false;
                            }
                        } while (flag);
                        //deja la i en el siguiente punto válido
                        if (invalid_index.Count != 0)
                        {
                            i = (int)invalid_index[invalid_index.Count - 1];
                            //se eliminan puntos inválidos
                            for (int j = 0; j < (invalid_index.Count - 1); j++)
                            {
                                temp_pol[(int)invalid_index[j], 0] = -1;
                                temp_pol[(int)invalid_index[j], 1] = -1;
                                temp_xy[(int)invalid_index[j], 0] = -1;
                                temp_xy[(int)invalid_index[j], 1] = -1;
                            }
                            //borra índices inválidos
```

```
                        invalid_index.Clear();
                    }
                }
            }
        }


        //ahora ya se tiene los puntos proyectados válidos
        //Se trasladan a arreglo de tamaño de escaneado
        indexProy.Clear();
        indexGlob.Clear();
        for (int i = 0; i < (polar.Length / 2); i++)
        {
            if (temp_pol[i, 0] != -1)
            {
                int ind = (int)((polar[i, 1] * 228.0) / 240);
                if (ind >= 228)
                {
                    ind = 227;
                }
                if (ind < 0)
                    ind = 0;
                proy_xy[ind, 0] = temp_xy[i, 0];
                proy_xy[ind, 1] = temp_xy[i, 1];
                proy_pol[ind, 0] = temp_pol[i, 0];
                proy_pol[ind, 1] = temp_pol[i, 1];
                indexProy.Add(ind); //agrega índice en proyección
                indexGlob.Add(i); //agrega índice en mapa global
            }
        }
    }

    //ALGORITMO DE ALINEAMIENTO Y COINCIDENCIA
    private void point_correspondence_matching()
    {

        //closest point rule
        int corresp = 0;
        double dist_min = 1000.0;
        double dista = 0.0;
        double[] dist_par = new double[228];
        //se identifica punto en proyección con distancia mínima
        // y se asigna como pareja a punto de escaneado nuevo
        for (int i = 0; i < 228; i++)
        {
            corresp = 0;
            dist_min = 1000.0;
            dista = 0.0;

            if (local_polar[i,0] != -1)
            {
                for (int j = 0; j < 228; j++)
                {
                    if (proy_pol[j, 0] != -1)
                    {
                        //calcula distancia entre puntos
                        dista = distancia2(local_xy[i, 0], local_xy[i, 1],
                                    proy_xy[j, 0], proy_xy[j, 1]);
                        if (dista <= dist_min)
                        {
                            //si es menor a la mínima, la reemplaza
                            corresp = j;
                            dist_min = dista;
                        }
                    }
                }
                //cuando ya se encontró el punto con la distancia mínima,
                // se guarda su índice en P1
                P1[i] = corresp;
```

```
                dist_par[i] = dist_min;

                //Se elimina puntos fuera de rango angular Bw
                if (Math.Abs((i * 240.0) / 228 - proy_pol[P1[i], 1]) > Bw)
                {

                    for (int a = 0; a < indexProy.Count; a++)
                    {
                        if ((int)indexProy[a] == P1[i])
                        {
                            indexProy.RemoveAt(a);
                            indexGlob.RemoveAt(a); //elimina de arreglo de omisión
                        }
                    }
                    P1[i] = -1;
                    dist_par[i] = -1;
                }
            }
            else
            {
                P1[i] = -1;
                dist_par[i] = -1;
            }
        }

        //Ahora se eliminan puntos que tienen al mismo correspondiente asignado,
        // la relación tiene que ser 1 a 1
        for (int k = 0; k < 228; k++)
        {
            if (P1[k] != -1)
            {
                for (int l = 0; l < 228; l++)
                {
                    //si tienen la misma pareja (y no es el mismo punto)
                    if (P1[k] == P1[l] && k != l)
                    {
                        //se deja solo el de distancia menor
                        if (dist_par[k] < dist_par[l])
                        {
                            //borra punto de arreglo de omisión
                            for (int a = 0; a < indexProy.Count; a++)
                            {
                                if ((int)indexProy[a] == P1[l])
                                {
                                    //elimina de arreglo de omisión
                                    indexProy.RemoveAt(a);
                                    indexGlob.RemoveAt(a);
                                }
                            }
                            dist_par[l] = -1;
                            P1[l] = -1;
                        }
                    }
                }
            }
        }

        //ahora se guardan distancias de puntos válidos
        ArrayList dist_m = new ArrayList();
        for (int i = 0; i < 228; i++)
        {
            if (dist_par[i] != -1)//si es válido, agrega distancia a arreglo
                dist_m.Add(dist_par[i]);
        }

        //Se eliminan los puntos que esten más alla de un treshold Br
        // se determina un límite Br para determinar parejas de puntos validas,
        // de tal forma que ||P'|-|P||<= Br
        //para esto, se elije Br/10
```

```
            double Br = 0;
            Br = 4096 / 10; //r/10
            //y se procede a eliminar los datos mas lejanos al límite
            for (int l = 0; l < P1.Length; l++)
            {
                //si el punto tiene pareja
                if (P1[l] != -1)
                {
                    //y si la distancia entre los puntos es mayor al threshold
                    if (distancia2(local_xy[l, 0], local_xy[l, 1],
                        proy_xy[P1[l], 0], proy_xy[P1[l], 1]) > Br)
                    {
                        //y se elimina punto de arreglo de omisión
                        for (int a = 0; a < indexProy.Count; a++)
                        {
                            if ((int)indexProy[a] == P1[l])
                            {
                                //elimina de arreglo de omisión
                                indexProy.RemoveAt(a);
                                indexGlob.RemoveAt(a);
                            }
                        }
                        P1[l] = -1; //se borra el punto
                    }
                }
            }
            //ahora ya se tiene el conjunto de puntos correspondientes
            // por la regla de punto mas cercano modificada
            //P1 guarda indices de puntos correspondientes en Sref.
            // P1[a] = b. a es el indice de Snew, b es el indice de Sref

            //Se procede a calcular la solución de mínimos cuadrados (w1, T1)
            // a partir de los pares correspondientes con P1

            closed_Form_PbM1();
        }

        //forma cerrada de solución de mínimos cuadrados
        private void closed_Form_PbM1()
        {
            double x_m = 0.0, y_m = 0.0, x_m1 = 0.0, y_m1 = 0.0;
            double Sxx1 = 0.0, Syy1 = 0.0, Sxy1 = 0.0, Syx1 = 0.0;

            int cuenta1 = 0;
            //se calculan primeros parámetros
            for (int a = 0; a < 228; a++)
            {
                if (local_polar[a,0] != -1 && P1[a] != -1)
                {
                    x_m += local_xy[a, 0];
                    y_m += local_xy[a, 1];
                    x_m1 += proy_xy[P1[a], 0];
                    y_m1 += proy_xy[P1[a], 1];
                    cuenta1++;
                }
            }
            x_m = x_m / cuenta1;
            y_m = y_m / cuenta1;
            x_m1 = x_m1 / cuenta1;
            y_m1 = y_m1 / cuenta1;

            //se calcula segundo set de parámetros
            for (int a = 0; a < 228; a++)
            {
                if (local_polar[a,0] != -1 && P1[a] != -1)
                {
                    Sxx1 += (local_xy[a, 0] - x_m) *
                        (proy_xy[P1[a], 0] - x_m1);
                    Syy1 += (local_xy[a, 1] - y_m) *
```

```
                    (proy_xy[P1[a], 1] - y_m1);
                Sxy1 += (local_xy[a, 0] - x_m) *
                    (proy_xy[P1[a], 1] - y_m1);
                Syx1 += (local_xy[a, 1] - y_m) *
                    (proy_xy[P1[a], 0] - x_m1);
            }
        }

        //con todos los parámetros necesarios,
        // se encuentra rotación y traslación
        w1 = Math.Atan(((Sxy1 - Syx1) / (Sxx1 + Syy1)));
        Tx1 = x_m1 - (x_m * Math.Cos(w1) - y_m * Math.Sin(w1));
        Ty1 = y_m1 - (x_m * Math.Sin(w1) + y_m * Math.Cos(w1));
    }

    //Se aplican resultados de algoritmo de alineamiento
    private void apply_transform(double ang, double TrX, double TrY)
    {
        //transformación angular en grados
        double angDegree = ((ang * 180.0) / Math.PI);
        //se crean arreglos temporales para almacenar puntos transformados
        float[,] temp_xy = new float[(mapa_polar.Length / 2), 2];
        float[,] temp_pol = new float[(mapa_polar.Length / 2), 2];

        //se aplica transformación polar
        for (int i = 0; i < (mapa_polar.Length / 2); i++)
        {
            if (mapa_polar[i, 0] != -1 )
            {
                temp_pol[i, 0] = mapa_polar[i, 0];
                temp_pol[i, 1] = (float)(mapa_polar[i, 1] - angDegree);
                if (temp_pol[i, 1] < 0)
                {
                    temp_pol[i, 1] = 360 + temp_pol[i, 1];
                }
                if (temp_pol[i, 1] >= 360)
                {
                    temp_pol[i, 1] = (temp_pol[i, 1] - 360);
                }
                temp_xy[i, 0] = (float)(temp_pol[i, 0] *
                            Math.Cos((temp_pol[i, 1] * Math.PI) / 180));
                temp_xy[i, 1] = (float)(temp_pol[i, 0] *
                            Math.Sin((temp_pol[i, 1] * Math.PI) / 180));
            }
            else
            {
                temp_pol[i, 0] = -1;
                temp_pol[i, 1] = -1;
                temp_xy[i, 0] = -1;
                temp_xy[i, 1] = -1;
            }
        }

        //Ahora se aplica transformación horizontal TrX y vertical TrY
        for (int i = 0; i < (mapa_polar.Length / 2); i++)
        {
            if (mapa_polar[i, 0] != -1 && mapa_polar[i, 1] != -1)
            {
                temp_xy[i, 0] = (float)(temp_xy[i, 0] - TrX);
                temp_xy[i, 1] = (float)(temp_xy[i, 1] - TrY);
                //distancia
                temp_pol[i, 0] =
                    (float)distancia2(temp_xy[i, 0], temp_xy[i, 1], 0, 0);
                //ángulo
                temp_pol[i, 1] =
                    (float)(angle(0, 0, temp_xy[i, 0], temp_xy[i, 1]) * 180 / Math.PI);
            }
            else
            {
```

```
                temp_pol[i, 0] = -1;
                temp_pol[i, 1] = -1;
                temp_xy[i, 0] = -1;
                temp_xy[i, 1] = -1;
            }
        }

        //se guardan valores en mapa global
        for (int i = 0; i < (mapa_polar.Length / 2); i++)
        {
            mapa_polar[i, 0] = temp_pol[i, 0];
            mapa_polar[i, 1] = temp_pol[i, 1];
            mapa_xy[i, 0] = temp_xy[i, 0];
            mapa_xy[i, 1] = temp_xy[i, 1];

        }
    }

    //aplica transformación a mapa local anterior
    private void apply_transform_ante(double ang, double TrX, double TrY)
    {
        //transformacion angular en grados
        double angDegree = ((ang * 180.0) / Math.PI);
        //se crean arreglos temporales para almacenar puntos transformados
        float[,] temp_xy = new float[228, 2];
        float[,] temp_pol = new float[228, 2];

        //se aplica transformación polar
        for (int i = 0; i < 228; i++)
        {
            if (ante_polar[i, 0] != -1)
            {
                temp_pol[i, 0] = ante_polar[i, 0];
                temp_pol[i, 1] = (float)(ante_polar[i, 1] - angDegree);
                if (temp_pol[i, 1] < 0)
                {
                    temp_pol[i, 1] = 360 + temp_pol[i, 1];
                }
                if (temp_pol[i, 1] >= 360)
                {
                    temp_pol[i, 1] = (temp_pol[i, 1] - 360);
                }
                temp_xy[i, 0] = (float)(temp_pol[i, 0] *
                            Math.Cos((temp_pol[i, 1] * Math.PI) / 180));
                temp_xy[i, 1] = (float)(temp_pol[i, 0] *
                            Math.Sin((temp_pol[i, 1] * Math.PI) / 180));
            }
            else
            {
                temp_pol[i, 0] = -1;
                temp_pol[i, 1] = -1;
                temp_xy[i, 0] = -1;
                temp_xy[i, 1] = -1;
            }
        }

        //Ahora se aplica transformación horizontal TrX y vertical TrY
        for (int i = 0; i < 228; i++)
        {
            if (ante_polar[i, 0] != -1 && ante_polar[i, 1] != -1)
            {
                temp_xy[i, 0] = (float)(temp_xy[i, 0] - TrX);
                temp_xy[i, 1] = (float)(temp_xy[i, 1] - TrY);
                temp_pol[i, 0] =
                    (float)distancia2(temp_xy[i, 0], temp_xy[i, 1], 0, 0);
                temp_pol[i, 1] =
                    (float)(angle(0, 0, temp_xy[i, 0], temp_xy[i, 1]) * 180 / Math.PI);
            }
            else
```

```
                {
                    temp_pol[i, 0] = -1;
                    temp_pol[i, 1] = -1;
                    temp_xy[i, 0] = -1;
                    temp_xy[i, 1] = -1;
                }
            }

            //se guardan valores en mapa local anterior
            for (int i = 0; i < 228; i++)
            {
                ante_polar[i, 0] = temp_pol[i, 0];
                ante_polar[i, 1] = temp_pol[i, 1];
                ante_xy[i, 0] = temp_xy[i, 0];
                ante_xy[i, 1] = temp_xy[i, 1];

            }
        }

        //METODOS GENERALES AUXILIARES
        /*-------------------------------------------------------------------------*/

        //metodo para revisar que trayectoria no atraviese objetos
        private bool check_path(float x_a, float y_a)
        {
            bool checkk = true;
            //primero se obtiene ángulo de siguiente posición respecto a 0 de robot
            double ang = 0.0;
            ang = angle(0, 0, x_a, y_a);
            //se convierte ángulo a grados
            ang = ang * 180 / Math.PI;

            //si el ángulo esta fuera del alcanze del sensor
            if (ang >= 240)
            {
                checkk = false;
            }
            else
            {
                //se encuentra el dato que corresponde a ese ángulo
                double posi_ = (228.0 * ang) / 240.0;
                 int posi = (int)posi_;
                //Se revisa mapa global para identificar validez de ruta
                for (int i = 0; i < mapa_polar.Length / 2; i++)
                {
                    double posimap_ = (228.0 * mapa_polar[i,1]) / 240.0;
                    int posimap = (int)posimap_;
                    if ((posimap + 1)>= posi && posi>=(posimap - 1))
                    {
                        //luego se verifica mapa local (arreglo dist)
                        // para revisar que no haya muro en el camino
                        //si la distancia es menor al valor máximo del sensor,
                        // significa que hay muro
                        if ((mapa_polar[i,0] < 4095 ||
                            (int)distancia2(0,0, x_a, y_a) > mapa_polar[i,0]) &&
                            mapa_polar[i,0] != -1)
                            checkk = false;
                    }
                }
            }
            return checkk;
        }

        //método para calcular ángulo entre dos puntos
        public double angle(double px1, double py1, double px2, double py2)
        {
            //se obtiene dif. en x y y entre los puntos
            double pxRes = px2 - px1;
            double pyRes = py2 - py1;
```

```
            double angle = 0.0;

            // se calula el ángulo
            if (pxRes == 0.0)
            {
                if (pyRes == 0.0)
                    angle = 0.0;
                else if (pyRes > 0.0)
                    angle = System.Math.PI / 2.0;
                else
                    angle = System.Math.PI * 3.0 / 2.0;
            }
            else if (pyRes == 0.0)
            {
                if (pxRes > 0.0)
                    angle = 0.0;
                else
                    angle = System.Math.PI;
            }
            else
            {
                if (pxRes < 0.0)
                    angle = System.Math.Atan(pyRes / pxRes) + System.Math.PI;
                else if (pyRes < 0.0) angle =
                    System.Math.Atan(pyRes / pxRes) + (2 * System.Math.PI);
                else
                    angle = System.Math.Atan(pyRes / pxRes);
            }
            //se devuelve el valor calculado
            return angle;
        }

        //calcula distancia entre posición actual y siguiente
        private double distancia2(double x_o, double y_o, double x_f, double y_f)
        {
            double dist2 = 0;
            dist2 = Math.Sqrt(Math.Pow((x_f - x_o), 2) + Math.Pow((y_f - y_o), 2));

            return dist2;
        }
    }
}
```

D. VISUALIZACIÓN EN 3D

1. CÓDIGO DEL PROGRAMA CON MODELOS 3D REALIZADO UTILIZANDO LAS LIBRERÍAS XNA DE DIRECTX

a. Clase principal

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using System.IO;

namespace mapeo3D_v3
{
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;
        public Camara camara1 { get; protected set; }
        public Camara camara2 { get; protected set; }
        public Camara currentDrawingCamera { get; protected set; }

        List<float> puntosx = new List<float>();
        List<float> puntosz = new List<float>();

        ModelManager modelManager;
        VertexPositionColor[] verts;
        VertexBuffer vertexBuffer;
        BasicEffect effect;

        float posx, posz;
        Color color1, color2;

        int actualizarNum = 8;
        int cuentaActualizar = 0;
        int angulo = 0;


        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";

            //pantalla completa
            //graphics.PreferredBackBufferWidth = 1280;
            // graphics.PreferredBackBufferHeight = 1280;
            // graphics.IsFullScreen = true;
        }
```

```
        protected override void Initialize()
        {
            //se inicializan las vistas de la cámara
            Viewport view1 = GraphicsDevice.Viewport;
            Viewport view2 = GraphicsDevice.Viewport;

            //se dan los tamaños a cada cámara
            view2.Width = view1.Width / 4;
            view2.Height = view2.Width;
            view2.X = view1.Width - view2.Width;
            view2.Y = view1.Height - view2.Width;

            //cámara utilizada para mapa 3D
            camara1 = new Camara(this, new Vector3(0, 500, 50), new Vector3(0,
500, 0), Vector3.Up, view1, false);
            Components.Add(camara1);

            //cámara utilizada para mapeo de planta
            camara2 = new Camara(this, new Vector3(0, 2000, 0), new Vector3(0, 0,
0), Vector3.Forward, view2, true);
            Components.Add(camara2);

            //clase para graficar los modelos 3D
            modelManager = new ModelManager(this);
            Components.Add(modelManager);

            base.Initialize();
        }


        //método para inicializar las figuras primitivas que se utilizan
        protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);
            List<VertexPositionColor> vertices = new List<VertexPositionColor>();

            //se inicializa el arreglo de vertices para la posición de la cámara
            verts = new VertexPositionColor[3];

            //se inicializa el efecto basico para graficar primitivas
            effect = new BasicEffect(GraphicsDevice);
            //set cullmode to none
            RasterizerState rs = new RasterizerState();
            rs.CullMode = CullMode.None;
            GraphicsDevice.RasterizerState = rs;

        }

        protected override void UnloadContent() { }

        //se actualizan las vistas de la cámara, y las figuras 3D
        protected override void Update(GameTime gameTime)
        {
            if (Keyboard.GetState().IsKeyDown(Keys.Escape))
                this.Exit();
```

```
            //se actualizan la informacón de modelos 3D, cada cierto tiempo
            cuentaActualizar++;
            if (cuentaActualizar == actualizarNum)
            {
                modelManager.changeContent();
                cuentaActualizar = 0;
            }

            //movimientos de las cámaras
            //////////////CAMARA 1/////////////////////////////////////
            //adelante y atras
            if (Keyboard.GetState().IsKeyDown(Keys.W))
            {
                camara1.adelanteAtras(true, 10);
            }
            if (Keyboard.GetState().IsKeyDown(Keys.S))
            {
                camara1.adelanteAtras(false, 10);
            }

            //a los lados
            if (Keyboard.GetState().IsKeyDown(Keys.A))
            {
                camara1.derechaIzquierda(false, 10);
            }
            if (Keyboard.GetState().IsKeyDown(Keys.D))
            {
                camara1.derechaIzquierda(true, 10);
            }

            //////////////CAMARA 2/////////////////////////////////////
            //adelante y atras
            if (Keyboard.GetState().IsKeyDown(Keys.I))
                camara2.arribaAbajo(false, 50);
            if (Keyboard.GetState().IsKeyDown(Keys.K))
                camara2.arribaAbajo(true, 50);

            //a los lados
            if (Keyboard.GetState().IsKeyDown(Keys.J))
                camara2.derechaIzquierda(false, 100);
            if (Keyboard.GetState().IsKeyDown(Keys.L))
                camara2.derechaIzquierda(true, 100);

            //rotacion YAW
            if (Keyboard.GetState().IsKeyDown(Keys.Right))
            {
                camara1.giro(false);
                angulo++;
            }
            if (Keyboard.GetState().IsKeyDown(Keys.Left))
            {
                camara1.giro(true);
                angulo--;
            }

            //se cambia la posición del triángulo que indica donde se encuentra la
cámara en el mapa.
            cambiarPosActual(camara1.cameraPosition.X, camara1.cameraPosition.Z);
```

```
            base.Update(gameTime);
        }

        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.Black);

            ///VIEWPORT 1
            ///se grafica sobre la cámara uno, la vista 3D
            currentDrawingCamera = camara1;
            GraphicsDevice.Viewport = camara1.viewport;
            effect.World = Matrix.Identity;
            effect.View = camara1.view;
            effect.Projection = camara1.projection;
            effect.VertexColorEnabled = true;
            foreach (EffectPass pass in effect.CurrentTechnique.Passes)
            {
                pass.Apply();

GraphicsDevice.DrawUserPrimitives<VertexPositionColor>(PrimitiveType.TriangleList,
verts, 0, 1);//.TriangleStrip, verts, 0, 2);
            }
            base.Draw(gameTime);


            ///VIEWPORT 2, se grafica sobre la cámara 2, la vista de planta
            currentDrawingCamera = camara2;
            GraphicsDevice.Viewport = camara2.viewport;
            effect.World = Matrix.Identity;
            effect.View = camara2.view;
            effect.Projection = camara2.projection;
            effect.VertexColorEnabled = true;
            foreach (EffectPass pass in effect.CurrentTechnique.Passes)
            {
                pass.Apply();

GraphicsDevice.DrawUserPrimitives<VertexPositionColor>(PrimitiveType.TriangleList,
verts, 0, 1);//.TriangleStrip, verts, 0, 2);
            }
            base.Draw(gameTime);
        }

        //grafica un plano triangular a una altura de 1 metro en la posición
actual de la cámara
        public void cambiarPosActual(float x, float z)
        {
            posx = x;
            posz = z;
            color1 = Color.Red;
            color2 = Color.Pink;
            int cuenta = 0;
            int ancho = 200;

            float tempx = ancho * (float)Math.Cos(MathHelper.ToRadians(angulo));
            float tempz = ancho * (float)Math.Sin(MathHelper.ToRadians(angulo));
            verts[cuenta] = new VertexPositionColor(new Vector3(posx + tempx,
1000, posz + tempz), Color.Yellow);
```

```
            cuenta++;
            tempx = ancho * (float)Math.Cos(MathHelper.ToRadians(angulo + 120));
            tempz = ancho * (float)Math.Sin(MathHelper.ToRadians(angulo + 120));
            verts[cuenta] = new VertexPositionColor(new Vector3(posx + tempx,
1000, posz + tempz), Color.Yellow);

            cuenta++;
            tempx = ancho * (float)Math.Cos(MathHelper.ToRadians(angulo - 120));
            tempz = ancho * (float)Math.Sin(MathHelper.ToRadians(angulo - 120));
            verts[cuenta] = new VertexPositionColor(new Vector3(posx + tempx,
1000, posz + tempz), Color.Red);

        }
    }
}
```

b. Componente cámara (Camara.cs)

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace mapeo3D_v3
{
    public class Camara : Microsoft.Xna.Framework.GameComponent
    {
        public Matrix view { get; protected set; }
        public Matrix projection { get; protected set; }
        public Vector3 cameraPosition { get; protected set; }
        Vector3 cameraDirection;
        Vector3 cameraUp;
        float speed = 10;
        int angle = 0;

        public Viewport viewport { get; set; }

        //método para inicializar la cámara
        public Camara(Game game, Vector3 pos, Vector3 target, Vector3 up,
Viewport viewport, bool orthographic)
            : base(game)
        {
            //se inicializan las principales propiedades de la cámara
            cameraPosition = pos;
            cameraDirection = target - pos;
```

```
            cameraDirection.Normalize();
            cameraUp = up;
            //se crea la matriz de vista
            CreateLookAt();

            //si es una vista ortográfica se crea otro tipo de projección
(ancho, alto, plano cercano, plano lejano)
            if (orthographic){
                projection = Matrix.CreateOrthographic(10000f, 10000f, 1,
100000);
            }
            //se define (campo de vista, aspecto, plano cercano, plano lejano)
            else
            {
                projection = Matrix.CreatePerspectiveFieldOfView(
                    MathHelper.PiOver4,
                    (float)Game.Window.ClientBounds.Width /
                    (float)Game.Window.ClientBounds.Height,
                    1, 100000);
            }

            this.viewport = viewport;
        }

        private void CreateLookAt() {
            view = Matrix.CreateLookAt(cameraPosition, cameraPosition +
cameraDirection, cameraUp);
        }

        //permite que se inicialice el componente de juego
        public override void Initialize()
        {
            base.Initialize();
        }

        //métodos para movimientos de la cámara
        public void adelanteAtras(bool adelante, float s) {
            if (adelante){
                cameraPosition += cameraDirection * s;
            }
            else {
                cameraPosition -= cameraDirection * s;
            }
        }

        public void derechaIzquierda(bool derecha, float s) {
            if (derecha){
                cameraPosition -= Vector3.Cross(cameraUp, cameraDirection) *
s;
            }
            else {
                cameraPosition += Vector3.Cross(cameraUp, cameraDirection) *
s;
            }
        }

        public void arribaAbajo(bool arriba, float s) {
            if (arriba){
```

```
                cameraPosition -= cameraUp * s;
            }
            else{
                cameraPosition += cameraUp * s;
            }
        }

        //giro tipo yaw
        public void giro(bool giroDerecha)
        {
            if (giroDerecha){
                angle=1;

            }
            else {
                angle=-1;

            }
            cameraDirection = Vector3.Transform(cameraDirection,
                Matrix.CreateFromAxisAngle(cameraUp, (float)(Math.PI/180
*angle)));
        }

        //método para actualizar las cámaras cosntantemente
        public override void Update(GameTime gameTime)
        {
            CreateLookAt();
            base.Update(gameTime);
        }
    }
}
```

c. Clase que maneja los modelos 3D (ModelManager.cs)

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using System.IO;

namespace mapeo3D_v3
{
    public class ModelManager : DrawableGameComponent
    {
        List<BasicModel> models = new List<BasicModel>();

        //dirección hacia archivo que contiene los puntos del mapa
        String path = "C:\\Users\\Mi
```

```
Equipo\\Dropbox\\mapeo3DSEEQ\\mapas\\mapaActual.txt";

        public ModelManager(Game game)
            : base(game){}

        //método para inicializar la clase
        public override void Initialize()
        {
            base.Initialize();
        }

        //método para cargar el archivo al iniciar el programa
        protected override void LoadContent()
        {
            //se utiliza la clase de PositionModel para posicionar cada modelo
3D
            models.Add(new
PositionModel(Game.Content.Load<Model>(@"models\Ground"), new Vector3(0, 0,
0)));
            models.Add(new
PositionModel(Game.Content.Load<Model>(@"models\techo"), new Vector3(0, 900,
0)));

            Boolean seguir = true;
            String line = "";

            List<VertexPositionColor> vertices = new
List<VertexPositionColor>();

            //se accede al archivo de texto
            StreamReader sr = new StreamReader(path);
            while (seguir)
            {
                line = sr.ReadLine();
                if (line != null)
                {
                    string[] puntos = line.Split(',');
                    if (puntos.Length > 1)
                    {
                        float x = (float)Convert.ToDouble(puntos[0]) * 1;
                        float z = (float)Convert.ToDouble(puntos[1]) * 1;
                        //se agrega cada modelo en cada posición encontrada en
el archivo
                        models.Add(new
PositionModel(Game.Content.Load<Model>(@"models\columna100x100"), new
Vector3(x, 0, z)));
                    }

                }
                else
                {
                    seguir = false;
                }

            }
            sr.Close();

            base.LoadContent();
```

```
        }

        public override void Update(GameTime gameTime){
            base.Update(gameTime);
        }

        //método para graficar cada modelo, en la cámara correspondiente
        public override void Draw(GameTime gameTime)
        {
            foreach (BasicModel bm in models) {
                bm.Draw(((Game1)Game).currentDrawingCamera);
            }
            base.Draw(gameTime);
        }

        //método para recargar todos los modelos
        public void changeContent() {
            base.UnloadContent();
            models.Clear();
            models.Add(new
PositionModel(Game.Content.Load<Model>(@"models\Ground"), new Vector3(0, 0,
0)));
            models.Add(new
PositionModel(Game.Content.Load<Model>(@"models\techo"), new Vector3(0, 900,
0)));

            Boolean seguir = true;
            String line = "";

            List<VertexPositionColor> vertices = new
List<VertexPositionColor>();

            StreamReader sr = new StreamReader(path);
            while (seguir)
            {
                line = sr.ReadLine();
                if (line != null)
                {
                    string[] puntos = line.Split(',');
                    if (puntos.Length > 1)
                    {
                        float x = (float)Convert.ToDouble(puntos[0]) * 1;
                        float z = (float)Convert.ToDouble(puntos[1]) * 1;

                        models.Add(new
PositionModel(Game.Content.Load<Model>(@"models\columna100x100"), new
Vector3(x, 0, z)));
                    }

                }
                else
                {
                    seguir = false;
                }

            }
            sr.Close();
```

```
            base.LoadContent();
        }

    }
}
```

d. Clase para posicionar modelo (PositionModel.cs)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace mapeo3D_v3
{
    class PositionModel: BasicModel
    {
        //traslada el mundo para dibujar el modelo 3D en la posición indicada.
        public PositionModel(Model m, Vector3 position) : base(m) {
            world = Matrix.CreateTranslation(position);
        }


        public override Matrix GetWorld()
        {
            return world;
        }

    }
}
```

e. Clase para dibujar un modelo básico (BasicModel.cs)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace mapeo3D_v3
{
    class BasicModel
    {
        public Model model { get; protected set; }
        protected Matrix world = Matrix.Identity;


    //se inicializa el modelo
```

```
    public BasicModel(Model m){
        model= m;
    }

    public virtual void Update(){}


    //se dibuja el modelo en la posición 0, 0, 0 del mundo
    public void Draw(Camara camara) {
        Matrix[] transforms = new Matrix[model.Bones.Count];
        model.CopyAbsoluteBoneTransformsTo(transforms);
        foreach (ModelMesh mesh in model.Meshes){
            foreach (BasicEffect be in mesh.Effects) {
                be.EnableDefaultLighting();
                be.Projection = camara.projection;
                be.View = camara.view;
                be.World = GetWorld() * mesh.ParentBone.Transform;
            }
            mesh.Draw();
        }
    }

    //se obtiene el mundo trasladado
    public virtual Matrix GetWorld() {
        return world;
    }

    }
}
```

E. INTEGRACIÓN DE MÓDULOS

Código de aplicación de megaproyecto SEEQ con todos los módulos integrados.

Nota: Este código utiliza la clase mapeo.cs cuyo código se muestra en los apéndices del módulo de mapeo y exploración.

1. APLICACIÓN DE INTEGRACIÓN

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Drawing2D;
using System.Collections;
using System.Drawing.Printing;
using System.IO;
using System.Runtime.InteropServices;

using System.Threading;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {

        private string echo = "", value = "", status = "", timestamp = "", sum
= "", templine = "";
        private string msj = "MS0044072503100\n";
        private string msj2 = "QT\n";
        private string strDatos = "";
        private byte [] datos;
        private int[] distancias;
        private int[] distanciasTemp;
        private int contdatos = 0, contdistancias = 0, i = 0, tempsum = 0;
        private int contscans = 0, numdatos = 0, contErrores = 0;
        private bool datoNuevo = false;

        //private int stepIni = 0, stepFin = 0;
        Mapeo mapa;
        public Bitmap myBitmap;
        Pen pen = new Pen(Color.Red);   //herramientas para dibujar
        Pen pen2 = new Pen(Color.Blue);
        //cuenta de repeticiones maxima
        private int cuentaLim = 0;

        private bool TxError = false;

        private byte[] Txdata = new byte[1];

        //inicializacion programa
        public Form1()
        {
            datos = new byte[64];
```

```
            distancias = new int[228];
            distanciasTemp = new int[228];

            InitializeComponent();
            Txdata[0] = (byte)144;

        //inicializacion de las constants con los numeric up/dwn

            Kp = (float)(controlp.Value) * 0.001 ;
            Ki = (float)(controli.Value) * 0.001;
            Kd = (float)(controld.Value) * 0.0001;
            Kdd = (float)(numericUpDown1.Value) * 0.0001;

        //inicializacion de array de distancias

            for (int i = 0; i < distancias.Length; i++)
            {
                distancias[i] = 0;
            }

            myBitmap = new Bitmap(pictureBox1.Size.Width,
                pictureBox1.Size.Height,
                System.Drawing.Imaging.PixelFormat.Format24bppRgb);
            myBitmap.MakeTransparent();



        }

        //boton para enviar codigo de peticion de escaneado
        private void button1_Click(object sender, EventArgs e)
        {
            if(HOKUYO.IsOpen){
                HOKUYO.Write(Txdata, 0, 1);
            }

            //---------------start Threading-----------------------------------
-----
            Thread mapear = new Thread(CosaMapas);
            mapear.Start();
        }

        //boton para abrir/cerrar comunicacion con puerto serial - comunicacion
RF
        private void button2_Click(object sender, EventArgs e)
        {
            if (HOKUYO.IsOpen == false)
            {
                HOKUYO.Open();
                button2.Text = "Close COM";
            }
            else
            {
                HOKUYO.Close();
                button2.Text = "Open COM";
            }
        }

        // METODOS Mario

        private void HOKUYO_DataReceived(object sender,
System.IO.Ports.SerialDataReceivedEventArgs e)
        {
```

```
            this.Invoke(new EventHandler(DoUpdate));
        }
        //lee datos actuales

        private void DoUpdate(object s, EventArgs e)
        {
            do
            {
                templine = HOKUYO.ReadLine();
                //richTextBox1.AppendText(templine + "\n");
                guardar_datos();
            }
            while (HOKUYO.BytesToRead > 0);
        }

        //guarda los datos
        private void guardar_datos()
        {
            if (ENDCMD() == true)
            {
                status = "";
                timestamp = "";
                contdatos = 0;
                i = 0;

                if (TxError)
                {
                    TxError = false;
                    contdistancias = 0;
                }
            }
            else
            {
                if (i == 0)
                {
                    echo = templine;
                }

                if (i == 1)
                {
                    status = templine;
                }

                if (i == 2)
                {
                    timestamp = templine;
                }

                if (i > 2)
                {
                    strDatos = templine;

                    if(!TxError)
                        procesar();
                }

                i++;
            }
        }
        //revisa la longitude de los datos

        private bool longitudDatos(string data)
        {
```

```
            numdatos = data.Length - 1;

            if ((numdatos == 64) || (numdatos == 8))
                return true;
            else
                return false;
        }
        //revisa la recepcion de datos

        private void procesar()
        {
            if (longitudDatos(strDatos) && checkSUM(strDatos))
            {
                for (int a = 0; a < strDatos.Length - 1; a++)
                {
                    if (a < datos.Length)
                    {
                        datos[a] = (byte)strDatos[a];
                        contdatos++;
                    }
                }

                getDistanceData(datos);
                contdatos = 0;
            }
            else
            {
                ManejarError();
            }
        }
        //manejo de errores

        private void ManejarError()
        {
            TxError = true;
            contErrores++;
            label22.Text = " " + contErrores;

            for (int i = 0; i < distancias.Length; i++)
            {
                //distancias[i] = 0;
                distanciasTemp[i] = 0;
            }

            if (longitudDatos(strDatos))
            {
                //richTextBox6.AppendText("ERROR de transmision - SCAN:" +
contscans + "\n");
                richTextBox6.AppendText("ERROR de transmision\n");
            }
            else
            {
                //richTextBox6.AppendText("ERROR de forma. Recibio: " +
numdatos + " en SCAN: " + contscans + "\n");
                richTextBox6.AppendText("ERROR de forma\n");
            }
        }
        //convierte a distancias

        private void getDistanceData(byte[] datosin)
        {
            for (int b = 0; b < contdatos / 2; b++)
            {
```

```
                if (contdistancias < distancias.Length)
                {
                    distanciasTemp[contdistancias] = decodechar("" +
(char)datosin[2 * b] + (char)datosin[2 * b + 1]);
                    if (distanciasTemp[contdistancias] < 20)
                        distanciasTemp[contdistancias] = -1;

                    contdistancias++;
                }
            }

            if (contdistancias == distanciasTemp.Length)
            {
                /*
                for (int i = 0; i < distanciasTemp.Length; i++)
                {
                    distancias[i] = distanciasTemp[i];
                }*/
                distancias = distanciasTemp;
                contdistancias = 0;
                contscans++;
                label24.Text = "" + contscans;
                //while (terminoMapa == false)
                //{ ;}
                    datoNuevo = true;  //recivio datos correctamente
            }
        }
        //revisa final comando

        private bool ENDCMD()
        {
            if (templine == "")
                return true;
            else
                return false;
        }

        private bool checkSUM(string data)
        {
            tempsum = 0;
            char SUM = data[data.Length - 1];

            for (int i = 0; i < data.Length - 1; i++)
            {
                tempsum += data[i];
            }

            sum = Convert.ToString((int)tempsum, 2).PadLeft(16, '0');
            sum = sum.Substring(10, 6);
            char c = (char)(Convert.ToInt16(sum, 2) + 0x30);
            //richTextBox6.AppendText(""+ c + "\n");

            if (SUM == c)
                return true;
            else
                return false;
        }
        //decodifica caracter

        private int decodechar(string chars)
        {
            value = "";
            for (int i = 0; i < chars.Length; i++)
```

```
            {
                value += Convert.ToString(chars[i] - 0x30, 2).PadLeft(6, '0');
            }
            return Convert.ToInt32(value, 2);
        }

        //METODOS SARAVIA

        int time = 0;
        float dT = 0;
        //bool terminoMapa = false;
        private void CosaMapas()
        {
            time = Environment.TickCount;
            finThread = false;
            do
            {
                //terminoMapa = false;
                while (!datoNuevo)
                {
                    Thread.Sleep(30);
                }
                datoNuevo = false;
                Saravia();
                //terminoMapa = true;
            }
            while (finThread == false);
        }
        float[] sp = new float[2]; //coordenadas con respecto a origen local de
siguiente posici
        private void Saravia()
        {
            if (mapa == null)
                {// primera vez
                    mapa = new Mapeo(distancias);
                    sp = mapa.sigPos(1);
                    cuentaLim = 1;
                    sp[0] = 0;
                    sp[1] = 0;
                    plotGlobal(sp[0], sp[1]);
                    MueveHelicoptero(sp[0], sp[1], 0, 0, (short)zeta.Value, 8);
//le indica a donde moverse

                }
                else
                {//Siguientes veces

                    mapa.localMap(distancias);
                    int x = 0;
                    int y = 0;
                    float[] tr = mapa.move(x, y);
                  //  sp[0] = 0;
                  //  sp[1] = 0;
                    float xtemp = sp[0];
                    sp[0] = (float)((xtemp - tr[0]) * Math.Cos(tr[2]) - (sp[1]
- tr[1]) * Math.Sin(tr[2]));
                    //sp[0] = (float)((sp[0]-tr[0])*Math.Cos(tr[2]));
                    //sp[1] = (float)((sp[1]-tr[1])*Math.Sin(tr[2]));
                    sp[1] = (float)((xtemp - tr[0]) * Math.Sin(tr[2]) + (sp[1]
- tr[1]) * Math.Cos(tr[2]));
                   // plotGlobal(sp[0], sp[1]);
                   MueveHelicoptero(sp[0], sp[1], 0, 0, (short)zeta.Value, 8);
                   // MueveHelicoptero((float)vX.Value, (float)vY.Value, tr[0],
```

```
tr[1], (short)zeta.Value, 8);
                    cuentaLim++;

                    if (cuentaLim <= 200)
                    {
                        if (cuentaLim % 10 == 0)
                        {
                            //recalcula siguiente mejor posici□
//--sp = mapa.sigPos(1);
                            plotGlobal(sp[0], sp[1]);

                        }
                        if (cuentaLim % 25 == 0)
                        {
                            //Actualiza archivo de texto
                            String txtfile = "";
                            for (int i = 0; i < mapa.mapa_xy.Length / 2; i++)
                            {
                                txtfile+=mapa.mapa_xy[i, 0] + ", " +
mapa.mapa_xy[i, 1] + "\n";
                            }
                            // escribe en archivo
                            System.IO.StreamWriter file = new
System.IO.StreamWriter(

"C:\\Users\\jose\\Dropbox\\mapeo3DSEEQ\\mapas\\mapaActual.txt");
                            file.WriteLine(txtfile);

                            file.Close();
                        }

                    }
                    else
                    { //termina escaneos predefinidos

                        plotGlobal(sp[0], sp[1]);
                         //HOKUYO.Write(Txdata, 0, 1); //toggle off hokuyo
                        finThread=true;
                    }

                }
        }

        bool finThread=false;


        //borra datos
        private void button3_Click(object sender, EventArgs e)
        {
            richTextBox1.Clear();
            richTextBox6.Clear();
        }

        //mapa para plotear mapa global actual
        double prop; //establece la proporcion para dibujar
        int xl0;
        int yl0;    //coordenadas para graficar
        private void plotGlobal(float xx, float yy)
        {
            xl0 = myBitmap.Size.Width / 2;
            yl0 = myBitmap.Size.Height / 2;
            prop = ((float)xl0)/MaxValue(mapa.mapa_polar);
            myBitmap = new Bitmap(pictureBox1.Size.Width,
```

```
                pictureBox1.Size.Height,
                System.Drawing.Imaging.PixelFormat.Format24bppRgb);
            myBitmap.MakeTransparent();
            double xl_ = 0.0, yl_ = 0.0;
            int maxData = mapa.mapa_polar.Length / 2;
            for (int i = 0; i < maxData; i++)
            {
             //   xl_ = xl0 + prop*mapa.mapa_xy[i, 0];
             //   yl_ = yl0 - prop*mapa.mapa_xy[i, 1]; //eje y esta invertido
                //xl = (int)xl_;
                //yl = (int)yl_;
                xl_ = xl0 +
prop*mapa.mapa_polar[i,0]*Math.Cos((mapa.mapa_polar[i, 1] - 30)*Math.PI/180.0);
                yl_ = yl0 - prop*mapa.mapa_polar[i, 0] *
Math.Sin((mapa.mapa_polar[i, 1] - 30)*Math.PI/180.0);


                if (mapa.mapa_polar[i, 0] != -1 && (xl_ > 0 && xl_ <
myBitmap.Size.Width) && (yl_ > 0 && yl_ < myBitmap.Size.Height))
                {
                    myBitmap.SetPixel((int)xl_, (int)yl_, Color.ForestGreen);
                }
            }

            double xl_2 = 0, yl_2 = 0; //coordenadas siguiente mejor posici
xl_2 = xl0 + prop * xx;
            yl_2 = yl0 - prop * yy; //eje y esta invertido

            //dibuja posicion del robot
            Graphics graphicsObj;
            graphicsObj = Graphics.FromImage(myBitmap);
            graphicsObj.DrawEllipse(pen, xl0 - 2, yl0 - 2, 4, 4); //dibuja
posicion actual de robot
            graphicsObj.DrawEllipse(pen2, (int)xl_2 - 2, (int)yl_2 - 2, 4, 4);
//dibuja siguiente posicion
            graphicsObj.DrawImage(myBitmap, 0, 0, myBitmap.Width,
myBitmap.Height);
            graphicsObj.Dispose();
            pictureBox1.Image = (Image)myBitmap;

        }


        //obtiene distancia maxima del mapa
        public float MaxValue(float[,] intArray)
        {
            float maxVal = intArray[0,0];
            for (int i = 1; i < intArray.Length/2; i++)
            {
                if (intArray[i,0] > maxVal)
                    maxVal = intArray[i,0];
            }
            return maxVal;
        }
       // para el helicoptero
        private void resetZ_Click(object sender, EventArgs e)
        {
            zeta.Value = 9;
            MueveHelicoptero(0, 0, 0, 0, (short)zeta.Value, 8);
        }

        //METODOS HUEVO Y SUAZO
        //w fijo en 8
```

```
        private void MueveHelicoptero(float Vx, float Vy, float dPx, float dPy,
short Z, short W)
        {
            short X;
            short Y;
            dT = (Environment.TickCount - time)*(float)0.001;
            time = Environment.TickCount;
            X = (short)PIDx(dPx, Vx);
            Y = (short)PIDy(dPy, Vy);
            mandaDatos(X, Y, Z, W);

        }
        double error_antDx = 0;
        double error_antPx = 0;
        double errorPx = 0;
        double errorIx = 0;
        double errorDx = 0;
        double errorDDx = 0;
        double errorX = 0;
        double Kp = 0, Ki = 0, Kd = 0, Kdd = 0;

       //control PID

        private double PIDx(float dPosx, float SPx)
        {
            //errorDx = errorDx * 0.5 + 0.5 * ((SPx - dPosx) - errorPx) / dT;
            errorPx = SPx - dPosx;
            errorIx += (errorPx * dT);
            errorDx = errorDx * 0.8 + 0.2 * (errorPx - error_antPx) / dT;
            errorDDx = errorDDx * 0.8 + 0.2 * (errorDx - error_antDx) / dT;
            error_antPx = errorPx;
            error_antDx = errorDx;
            errorX = (errorPx * Kp + errorIx * Ki + errorDx * Kd + errorDDx *
Kdd) +(float)(Xmanual.Value);

            if (errorIx > 20)
                errorIx = 20;
            if (errorIx < -20)
                errorIx = -20;
            if (errorX > 31)
                errorX = 31;
            if (errorX < -31)
                errorX = -31;
            return errorX;
        }
        double error_antDy = 0;
        double error_antPy = 0;
        double errorPy = 0;
        double errorIy = 0;
        double errorDy = 0;
        double errorDDy = 0;
        double errorY = 0;

       //control PID sobre un eje

        private double PIDy(float dPosy, float SPy)
        {
            //errorDy = errorDy * 0.5 + 0.5 * ((SPy - dPosy) - errorPy) / dT;
            errorPy = SPy - dPosy;
            errorIy += (errorPy * dT);
            errorDy = errorDy * 0.8 + 0.2 * (errorPy - error_antPy) / dT;
            errorDDy = errorDDy * 0.8 + 0.2 * (errorDy - error_antDy) / dT;
            error_antPy = errorPy;
```

```
            error_antDy = errorDy;
            errorY = (errorPy * Kp + errorIy * Ki + errorDy * Kd + errorDDy *
Kdd) + (float)(Ymanual.Value);

            if (errorIy > 20)
                errorIy = 20;
            if (errorIy < -20)
                errorIy = -20;
            if (errorY > 31)
                errorY = 31;
            if (errorY < -31)
                errorY = -31;
            return errorY;
        }
       //envoi y revision de datos por el COM
        private void mandaDatos(short x, short y, short z, short w)
        {
            if (x > 31)
                x = 31;
            if (x < -31)
                x = -31;

            if (y > 31)
                y = 31;
            if (y < -31)
                y = -31;

            if (w > 63)
                w = 63;
            if (w < 1)
                w = 1;

            if (z > 63)
                z = 63;
            if (z < 1)
                z = 1;


            byte[] buffer = new byte[4];
            buffer[0] = (byte)(x + 32);
            buffer[1] = (byte)(y + 32 + 64);
            buffer[2] = (byte)(w + 2 * 64);
            buffer[3] = (byte)(z + 3 * 64);

            //serialPort1.Write(buffer, 0, 4);
             HOKUYO.Write(buffer, 0, 4);
            //label18.Text = "" + buffer[0] + " " + buffer[1] + " " + buffer[2]
+ " " + buffer[3];
        }

       //Setea los valores de las constantes

        private void controlp_ValueChanged(object sender, EventArgs e)
        {
            Kp = (float)(controlp.Value) * 0.001;
        }

        private void controli_ValueChanged(object sender, EventArgs e)
        {
            Ki = (float)(controli.Value) * 0.001;

        }
```

```
        private void controld_ValueChanged(object sender, EventArgs e)
        {
            Kd = (float)(controld.Value) * 0.0001;

        }
        private void numericUpDown1_ValueChanged(object sender, EventArgs e)
        {
            Kdd = (float)(numericUpDown1.Value) * 0.0001;
        }

        private void zeta_ValueChanged(object sender, EventArgs e)
        {

        }

        private void resetM_Click(object sender, EventArgs e)
        {
            mapa = null;
            richTextBox1.Text = "";
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            label13.Text = "van: " + cuentaLim;
         //   label19.Text = "S.P. X=" + sp[0];
         //   label20.Text = "S.P. Y=" + sp[1];
            label19.Text = "X=" + errorX;
            label20.Text = "Y=" + errorY;

            if (finThread)
            {
                if (HOKUYO.IsOpen)
                {
                    HOKUYO.Write(Txdata, 0, 1);
                }
                if (HOKUYO.IsOpen)
                {
                    HOKUYO.Write(Txdata, 0, 1);
                }
                if (HOKUYO.IsOpen)
                {
                    HOKUYO.Write(Txdata, 0, 1);
                }
                //if (HOKUYO.IsOpen)
                //    HOKUYO.WriteLine(msj2);
                label13.Text = "fin";
                finThread = false;
            }
        }
    }
}
```

XVII. GLOSARIO

Acelerómetro: Sensor de aceleración. Utilizado para obtener el ángulo, midiendo la dirección del vector de gravedad.

ADC: Conversión análogo digital. Utilizado para la interfaz con los sensores de la IMU.

Aerodinámica: Es la rama de la mecánica de fluidos que estudia las acciones que aparecen sobre los cuerpos sólidos cuando existe un movimiento relativo entre éstos y el fluido que los baña, siendo éste último un gas y no un líquido.

Algoritmo: Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.

Arrastre: Es la fricción entre un objeto sólido y el fluido o gas por el que se mueve.

Autonomía: La cualidad o estado de auto gobernarse y ser independiente. [30]

Bancada: Base sobre la cual se monta un motor.

Catalizador: Sustancia que está presente en una reacción química en contacto físico con los reactivos, y acelera dicha reacción sin actuar en la misma.

Control Digital: Sistema controlador en el tiempo discreto.

Control PID: Control Proporcional Derivativo e Integral.

Cuadcóptero: Vehículo volador de 4 motores, con movimientos similares a los de un helicóptero.

C#: Lenguaje de programación moderno, desarrollado por Microsoft. [9]

Determinístico: Sistema con una respuesta o comportamiento predecible. [63]

DirectX: Interfaz de programación de Microsoft que proporciona a los programadores una forma estándar de acceder directamente a las funciones del hardware. [43]

Ductilidad: Propiedad que presentan ciertos materiales de deformarse sosteniblemente sin romperse, permitiendo obtener alambres o hilos.

Entropía: En la teoría de la información, se define a la entropía como la medición de la cantidad de información provista por una fuente de datos. [18]

Esfuerzo: Es la resistencia que ofrece un área unitaria del material del que está hecho un miembro para una carga aplicada externa.

Factor de seguridad: Margen de error de cada diseño, su magnitud es proporcional a la cantidad de desconocimiento implícito.

Filtro complementario o ponderado: Filtro utilizado para obtener un promedio de 2 lecturas, obteniendo una mejor estimación del resultado.

Filtro de ventana: Filtro con valores máximos y mínimos, saturando la señal de entrada.

Filtro pasa baja: Filtro que atenúa frecuencias altas.

Flexión: Tipo de deformación que presenta un elemento estructural alargado, en dirección perpendicular a su eje longitudinal.

Fluencia: Punto en el que un material dúctil continúa su deformación, sin incremento en la carga aplicada en un ensayo de tensión.

Flujo: Cantidad de masa o volumen por unidad de tiempo.

Framework: Estructura compuesta de partes diseñadas para encajar unas con otras. [48]

Frecuencia de corte: Frecuencia a la cual un sistema es atenuado 3 decibeles por debajo de la amplitud máxima.

Frecuencia natural: Es la frecuencia propia de un cuerpo o sistema por poseer elementos elásticos e inerciales.

Función de transferencia: Es un modelo matemático que a través de un cociente relaciona la respuesta de un sistema, a una señal de entrada o excitación.

Giroscopio: Sensor de velocidad angular.

GUI: Acrónimo para "*Graphical User Interface*". Interfaz gráfica de usuario.

Hardware: Elementos mecánicos, electrónicos, eléctricos y magnéticos que contiene un computador. [48]

Hz: Dimensional de la frecuencia, nombrada así en honor al científico alemán Henry Hertz.

IMU: Inertial Measure Unit. Unidad que comprende los acelerómetros y giroscopios.

Incertidumbre: Parámetro asociado con mediciones y cálculos que caracteriza la dispersión de valores que podrían ser atribuidos a la variable de medición [62]

Interfaz: Componente de una aplicación que permite el intercambio de información entre el usuario y el programa.

ISA: Instruction Set Arquitecture.

Lenguajes .NET: Entorno de desarrollo y ejecución de software, diseñado principalmente para Microsoft Windows. Incluye una librería extensa que soporta varios lenguajes de programación. [48]

Odometría: Técnicas de posicionamiento que emplean información de sensores para obtener una aproximación de la posición real de un sistema móvil, respecto a un sistema de referencia inicial. [46]

OpenGl (librería de gráficos abierta): Interfaz de programación para el hardware gráfico, que soporta acciones de renderizado y operaciones con imágenes. [48]

PDF: Acrónimo de "*non informative probability distribution function*". Es una función que describe la probabilidad de una variable aleatoria dados ciertos valores. [56]

PITCH: Ángulo sobre el eje Y.

PWM: Modulación de ancho de pulso.

Rango: Intervalo de valores permisibles delimitado por un máximo y un mínimo.

Renderización: Cálculo que hace el ordenador para generar una imagen desde un modelo. [48]

Resina: Es una secreción orgánica que producen muchas plantas, particularmente los árboles del tipo conífera. Es muy valorada por sus propiedades químicas y sus usos asociados, como por ejemplo la producción de barnices, adhesivos y aditivos alimenticios.

ROLL: Ángulo sobre el eje X.

SEEQ: Smart Enviroment Exploring Quadcopter. Siglas del Megaproyecto.

Síncrono: Que tiene intervalo de tiempo constante entre cada evento, literalmente quiere decir al mismo tiempo.

SLAM: Acrónimo para "*Simultaneous Localization and Mapping*". Problema de generación de mapa y localización simultáneos. [38]

Software: Programas utilizados para dirigir las operaciones de una computadora. [48]

Sustentación: Fuerza generada sobre un cuerpo que se desplaza a través de un fluido, de dirección perpendicular a la velocidad de la corriente incidente.

Trifásico: Que posee tres corrientes alternas desfasadas 120 grados.

Ultrasónico: Onda acústica imperceptible por el ser humano por estar en una frecuencia más alta a la que el oído puede captar. Su frecuencia límite se encuentra aproximadamente en 20kHz.

Vibración: Propagación de ondas elásticas produciendo deformaciones y tensiones sobre un medio continuo.

Xbee: Solución integrada que provee conectividad wireless de punto a punto entre dispositivos. [54]

XNA: Set de componentes de programa pre construidos que se pueden utilizar como parte de otros programas. [43]

YAW: Ángulo respecto el eje vertical.