Robótica Situada Aplicada al Control de Vuelo Autónomo de un Cuadricóptero

Diego Avila[1], Emiliano Lorusso[1], Sofia Fasce[1], Gustavo Pereira[1], Norberto Mazza[1], Juan Iribarren[1], Rodolfo Meda[1] ,Jorge Ierache[1]
Instituto de Sistemas Inteligentes y Enseñanza Experimental de la Robótica (ISIER)[1]

ISIER, Facultad de Informática Ciencias de la Comunicación y Técnicas Especiales
Universidad de Morón, Cabildo 134, (B1708JPD) Morón, Buenos Aires, Argentina
54 11 5627 200 int 189
jierache@yahoo.com.ar

Resumen. En este artículo se presenta los resultados iniciales obtenidos con del empleo de un mini cuadricóptero en el contexto de la robótica situada, en este orden se emplea visión externa para capturar el ambiente de actuación, conformado por cuatro puntos o checkpoint que el sistema debe sobrevolar en forma autónoma, se discute la implementación y se presentan los resultados de las pruebas realizadas con el cuadricóptero guiado a través de visión externa.

Palabras Claves: Robots autónomos, robótica, vehículos aéreos, visión externa.

1 Introducción

El uso de vehículos aéreos para realizar diferentes tipos de tareas es de especial interés en la robótica, y también en la robótica autónoma. La cantidad de parámetros que tiene que tener en cuenta a la hora de moverse por el ambiente es mayor a la de cualquier otro tipo de vehículo, así como también sus grados de libertad. Teniendo en cuenta esto, es mayor la complicación cuando el vehículo tiene muy pocos sensores y lo que percibe del ambiente es solamente una imagen. Dentro de éstas consideraciones nos propusimos experimentar con un robot desarrollado por un grupo originario de Suecia, integrado por Marcus Eliasson, Tobias Antonsson y Arnaud Taffanel [1], que nos brindaba todas las herramientas necesarias para desarrollar y experimentar teniendo en cuenta las consideraciones anteriores. El desarrollo de los experimentos realizados, se hicieron en el lenguaje C++ utilizando libcflie [2] y OpenCV [3] y en un entorno Linux (Ubuntu 12.04). El objetivo de este trabajo es explicar los problemas a los que nos enfrentamos a la hora de trabajar con el cuadricóptero, y las soluciones propuesta en el marco de un sistema situado con el empleo de visión externa.

2 Problema

En el campo de la robótica situada se encuentran distintas aplicaciones y competencias como lo es el fútbol de robots correspondiente a categorías físicas que reciben el ambiente del campo de juego desde una cámara suspendida, como ejemplo entre otros, tenemos la categoría F-180 de Robocup [4]. En este orden el presente trabajo plantea como problema generar un ambiente de robótica situada aplicada a robots aéreos, considerando el control autónomo de vuelo del mismo a partir de la información del ambiente provisto por una cámara suspendida. El sistema de visión debe permitir obtener la posición del Robot en vuelo como así también del resto de los elementos que conforman el mismo, en este caso elementos fijos conformados por cuatro checkpoint.

3 Solución Propuesta

El algoritmo de visión utilizado para el reconocimiento de los checkpoints y del robot, y para ubicarlo en el ambiente funciona de la manera descrita a continuación. En primer lugar, se trataron las imágenes de acuerdo a un rango de colores que eran únicos dentro del ambiente, y que representaban a los respectivos objetivos. Para esto, se trabajó con el modelo de color HSB/HSV (Hue, Saturation, Brightness o Value). Este espacio de colores, los representan combinando tres valores: el tono en sí (Hue o Tonalidad), la saturación (Saturation o Saturación) y el brillo (Brightness o Brillo). Una vez que se pasó la imagen a este espacio de color, se buscaron los rangos de colores que representan los objetivos (el cuadricóptero, los checkpoint).

En segundo lugar, se debieron filtrar las imágenes para eliminar el posible ruido. Para alcanzar esto, se debe reducir la amplitud de las variaciones de la imagen, reemplazando cada pixel por la media del valor de los píxeles de alrededor y así, las variaciones rápidas de intensidad pueden ser suavizadas y reemplazadas por una transición más gradual. OpenCV provee una función llamada GaussianBlur, que aplica el desenfoque gaussiano [5] y [6], y que tiene en cuenta el peso de los píxeles más cercanos. En tercer lugar, se debieron filtrar los bordes de la imagen para luego detectar formas en ella. Para realizar este paso, se utilizó el Algoritmo de Canny [7], implementado por OpenCV. Este algoritmo pasa por cuatro etapas a la hora de definir contornos en la imagen: el primer paso es la reducción de ruido mediante el desenfoque gaussiano; el segundo paso es encontrar el gradiente de intensidad de la imagen; el tercer paso es la remoción de los pixeles que no son considerados parte del borde; y el último paso es la histéresis aplicando dos umbrales, uno máximo y otro mínimo. Por último, fue necesario encontrar la forma designada como objetivo. En todas las pruebas, tanto el robot como los *checkpoints* fueron representados por medio de círculos y se los buscaron mediante la transformada de Hough [8] y [9]. Esta técnica permite detectar figuras como círculos, cuadrados, elipses y líneas rectas, dentro de la imagen. Dado que el círculo a detectar se mueve dentro de la imagen y que puede haber falsos positivos, se ideó un algoritmo para reducir el rango de error propio de la detección de un figura tan común. Para esto, se tiene en cuenta la posición de todos los círculos que se encuentran en la imagen y se compara su

posición con la posición del círculo detectado en el cuadro anterior del video, eligiendo el círculo detectado más cercano al círculo anterior, y descartando el resto de las detecciones. El control del robot cuadricóptero que se muestra en la figura 1, se implementó analizando las capturas de video, y corrigiendo la trayectoria del robot para mantenerlo estable cuando era posible. Fue posible corregir este desvío ya que teníamos una clara noción de la posición del robot y de a dónde se ubicaba el objetivo hacia donde debía ir. En este caso, utilizamos un algoritmo PID (Proportional, Integral, Derivative), el cual es un mecanismo de control con retroalimentación, para la corrección de los errores propios del traslado del vehículo. Este algoritmo calcula el error sobre los valores actuales, en relación al valor esperado, y luego se le aplica una corrección para mejorar el vuelo del cuadricoptero. Aplicamos PID en el cabeceo y el balanceo del robot, para determinar con precisión el ángulo y la velocidad a la hora de alcanzar el punto definido como objetivo.

Fig 1. Cuadricóptero con identificación de color.

Para que el algoritmo funcione correctamente, se deben definir tres constantes: una constante de proporcionalidad (Kp), una constante de integración (Ki) y una constante de derivación (Kd). Cada una de las constantes, influye de diferentes formas a la hora de corregir el error. La constante de proporcionalidad o Kp, representa el valor de ganancia del error; la constante de integración o Ki, representa la velocidad con la que se repite la acción de proporcionalidad; y la constante de derivación o Kd, representa el lapso de tiempo durante el cual se aplicará la acción de proporcionalidad. Estas constantes fueron definidas y refinadas empíricamente. Además se definió un valor máximo y mínimo para la variable de integración. En el caso del cabeceo, utilizamos los siguientes valores, Kp: 0.05, Ki: 0.00025, Kd: 0.25, Integración máxima: 250, Integración mínima: -100. En el caso del balanceo, Kp: 0.04, Ki: 0.00025, Kd: 0.25,

Integración máxima: 250, Integración mínima: -100. El algoritmo calcula el error en base al valor real del cabeceo o balanceo:

```
this->error = this->set_point - current_value;
```

luego calcula los valores para las variables de derivación, proporción e integración:

```
P_value = Kp * error;
float change = error - last_error;
if(error > 0)
      I_value = Integrator * Ki;
else
      I_value = (Integrator * Ki) * 0.5;
D_value = Kd * ( error - Derivator);
D_value = Kd * change;
Derivator = error;
Integrator = Integrator + error / 200;
```

y por último define un nuevo valor para corregir el error:

```
last_error = error;
last_value = current_value;
return P_value + I_value + D_value;
```

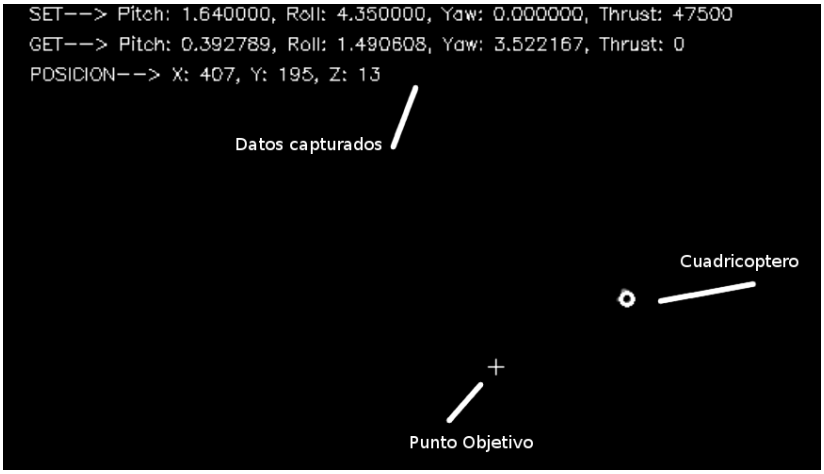

Fig 2. Análisis de la imagen obtenida con la cámara.

Estos valores se calculan por cada ciclo de procesamiento, teniendo en cuenta los valores al instante del cabeceo (pitch) y el balanceo (roll) del vehículo. Comandos para implementar la guiñada (yaw) no se aplicaron en esta etapa. La figura 2 muestra la imagen capturada desde la aplicación implementada empleando la librería libcflie la que muestra en un instante los datos capturados en relación a la posición del robot cuadricóptero en vuelo y los valores que se le envían al robot.

4 Pruebas realizadas y resultados obtenidos.

Los experimentos se basaron en la realización de tres pruebas conformados con un set de navegación que se repite en cada caso de prueba cuatro veces. Como métrica se estableció calcular la media de tiempo en el patrón de vuelo del robot y compararla con una media ideal de vuelo sobre la base de una velocidad de 0,3 m/seg, asumiendo un patrón ideal de vuelo sobrevolando cada checkpoint en su centro. El ambiente de pruebas fue iluminado específicamente a fin de evitar falsos positivos en la detección de objetos.

El modelo del ambiente utilizado en todas las pruebas se muestra en la figura 3.a. Se observa un campo de vuelo de 3 metros por 2 metros, definido por el ángulo de visión de la cámara, y ésta colocada en el centro del campo a 4 metros de altura. Dentro del campo, se situaron 4 objetivos a alcanzar por el vehículo, que eran identificados a través del análisis de la imagen capturada por la cámara, de colores diferentes e irrepetibles. El *checkpoint* "A", de color amarillo; el *checkpoint* "B" de color verde; el *checkpoint* "C" de color naranja; y el *checkpoint* "D" de color negro. Todos los objetos identificables, son únicos dentro del ambiente para lograr un correcto reconocimiento de los mismos.

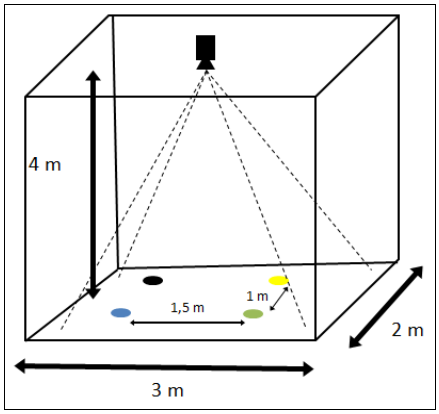

Fig 3.a Ambiente utilizado.

Las tres pruebas realizadas tuvieron características similares: en todas, el robot partió de una posición arbitraria en el ambiente y diferente de las anteriores,

realizando el patrón de vuelo A → B → C → D; para que se dé por concluido el *checkpoint* el robot debió permanecer al menos tres segundos en un radio de 10 cm del mismo.

Se tomó como base un tiempo ideal dentro del cual es aceptable el comportamiento del robot. Para definir el tiempo para cada tramo a recorrer, partimos del siguiente supuesto: como velocidad ideal de vuelo tomamos 0.3 m/seg considerando que el robot parte de un vuelo estacionario, hacia el siguiente *checkpoint* y cuando llega a éste, no se produce ningún desvío. Además, a este tiempo se le agregó un *delta* que incluye el tiempo que debe permanecer el robot en el *checkpoint* y un tiempo de 4 segundos para estabilizarse.

En el marco de las pruebas el despegue y la altura de vuelo no fue automático, sino que fue dirigida, en razón que en el alcance inicial de este trabajo se pretende ver el comportamiento del cuadricóptero en función de la información que le envía el sistema de visión externo.

Se observa en la Figura 3.b, una imagen real del escenario de pruebas con el robot en vuelo, para mejor proveer se publicaron videos de las mismas [10].

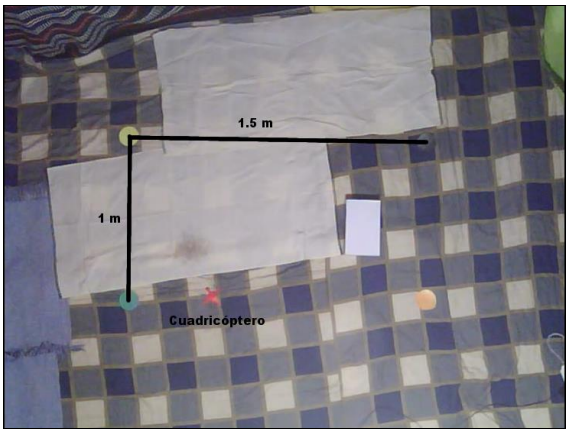

Fig 3.b Imagen del Ambiente real de pruebas utilizado.

4.1 Primer prueba

En esta prueba se colocó el vehículo en la mitad del segmento $\overline{AB}$. Desde el instante mismo en que el robot despega del suelo, se comenzó a cronometrar los tiempos de vuelo. Como se dijo anteriormente, el vehículo realizó el siguiente patrón de vuelo: primero el *checkpoint* A, situado a 50 cm del punto de despegue, y luego el *checkpoint* B, C y por último el *checkpoint* D. En la figura 4 se puede apreciar las distancias y el recorrido del robot. Los resultados de estas pruebas se detallan en la tabla 1.

Tabla 1. Resultados primer set de pruebas

Prueba 1 \ Checkpoint	**A**	**B**	**C**	**D**	**Tiempo de finalización**
Test a	15.52 s	3.61 s	6.59 s	9.29 s	35.01 segundos
Test b	15.04 s	6.69 s	14.32 s	17.25 s	53.3 segundos
Test c	14.54 s	4.25 s	12.34 s	10.6 s	41.73 segundos
Test d	14.7 s	5.15 s	6.85 s	11.13 s	37.33 segundos
Promedio	14.95 s	4.93 s	10.025 s	12.07 s	41.8425 segundos

El tiempo de vuelo promedio fue de 41.8425 segundos, contrastando con el tiempo ideal de vuelo para esta prueba: 33.26 segundos, lo que implica una diferencia del 25.8%.

4.2 Segunda prueba

En esta prueba se colocó el vehículo en el centro del campo de vuelo. Desde el instante mismo en que el robot despega del suelo, se comenzó a cronometrar los tiempos de vuelo. Como se dijo anteriormente, el vehículo realizó el siguiente patrón de vuelo: primero el *checkpoint* A situado a 90 cm del punto de despegue, y luego el *checkpoint* B, C y por último el *checkpoint* D. En la Figura 4 se puede apreciar las distancias y el recorrido del robot.

Los resultados de esta prueba se detallan en la tabla 2:

Tabla 2. Resultados primer set de pruebas.

Prueba 2 \ Checkpoint	**A**	**B**	**C**	**D**	**Tiempo de finalización**
Test a	17.12 s	3.3 s	6.34 s	8.85 s	35.61 segundos
Test b	16.76 s	3.42 s	5.86 s	9.89 s	35.93 segundos
Test c	19.24 s	5.53 s	7.8 s	9.16 s	41.73 segundos
Test d	15.2 s	4.15 s	6.15 s	8.76 s	34.26 segundos
Promedio	17.08 s	4.1 s	6.54 s	9.165 s	36.8825 segundos

El tiempo de vuelo promedio fue de 36.8825 segundos, contrastando con el tiempo ideal de vuelo para esta prueba: 35.6 segundos, lo que implica una diferencia del 3.6%.

4.3 Tercer Prueba

En esta prueba se colocó el vehículo en la mitad del segmento $\overline{DA}$. Desde el instante mismo en que el robot despega del suelo, se comenzó a cronometrar los tiempos de vuelo. Como se dijo anteriormente, el vehículo realizó el siguiente patrón de vuelo: primero el *checkpoint* A, el cual se situa a 75 cm del punto de despegue, y luego el *checkpoint* B, C y por último el *checkpoint* D. En la figura 4 se puede apreciar las distancias y el recorrido del robot. Los resultados de esta prueba se detallan en la siguiente tabla 3:

Tabla 3. Resultados del tercer set de pruebas

Prueba 3 \ Checkpoint	A	B	C	D	Tiempo de finalización
Test a	16.65 s	5.3 s	6.64 s	10.85 s	39.44 segundos
Test b	15.96 s	3.42 s	5.46 s	9.76 s	34.6 segundos
Test c	16.24 s	4.06 s	7.15 s	11.05 s	38.5 segundos
Test d	17.04 s	4.15 s	6.59 s	9.33 s	37.11 segundos
Promedio	16.47 s	4.23 s	6.46 s	10.25 s	37.4125 segundos

El tiempo de vuelo promedio fue de 37.4125 seg, contrastando con el tiempo ideal de vuelo para esta prueba 33.26 seg, lo que implica una diferencia del 12.49%.

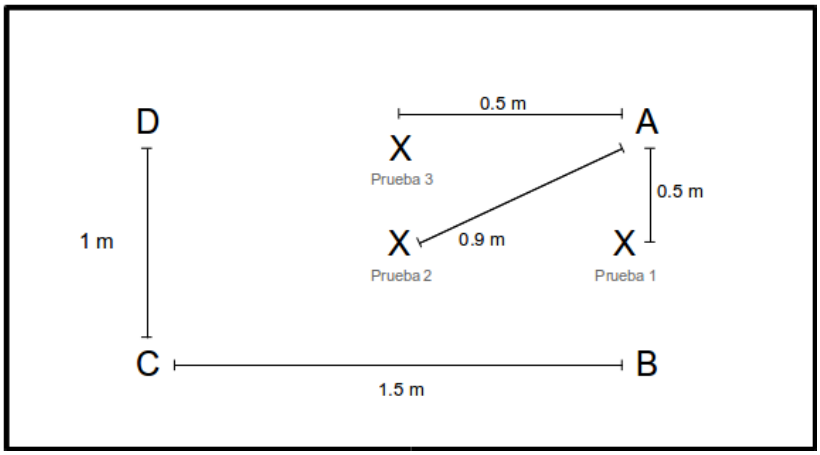

Fig 4. Diagrama con las posiciones de despegue del robot y los *checkpoints.*

4.4 Conclusiones de las pruebas

Se presenta en forma de síntesis una grafica con los tiempos promedios correspondientes a cada checkpoint en función de cada prueba realizada (figura 5). Como conclusión final, en base a las pruebas realizadas y a los resultados ideales esperados, entendemos que el 13.96%, en promedio, de diferencia entre unos y otros (figura 6) es producto del acondicionamiento, en vuelo, que debe hacer el robot a la hora de posicionarse en el objetivo. El principal desvío sucede a la hora del despegue, entre que el robot se desprende del suelo y acondiciona su vuelo hacia el primer checkpoint.

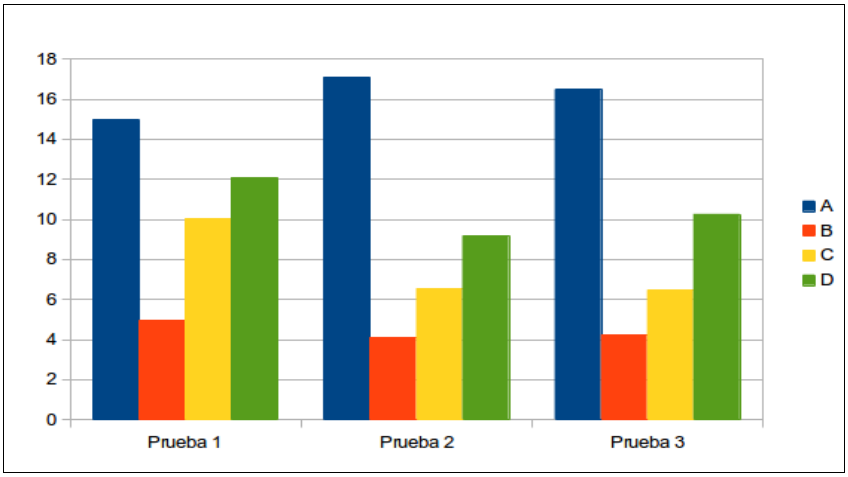

Fig 5. Tiempos promedios por cada prueba, por checkpoint

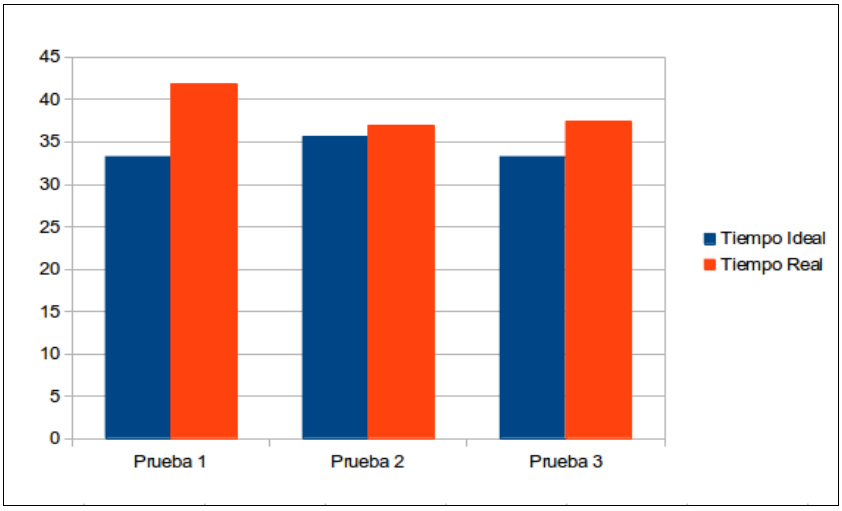

Fig 6. Comparación entre el tiempo ideal y el tiempo real en las pruebas

5 Conclusiones y futuras líneas de trabajo

En función de los resultados obtenidos a través de las pruebas del sistema de captura y procesamiento del ambiente de vuelo del cuadricóptero, podemos concluir que se cumplimentaron los distintos patrones de vuelo en forma aceptable en función de las diferencias promedio de tiempo total en relación a la métrica establecida de

vuelo ideal La futuras líneas de trabajo se centran en, el control de altura del cuadricóptero, en la versión aplicada no fue posible la incorporación de sensores extras para inferir la altura, en estas líneas de trabajo se explora el empleo de cuadricópteros de mayor porte, como así también el empleo de una cámara lateral para conformar la información del ambiente, que permitiera estimar la altura relativa del robot y como segunda opción, sugerimos la utilización de visión estereoscópica en lo alto, y de este modo reconocer la distancia a la que se encuentra el robot, en relación a las cámaras superiores.

En el marco de empleo de cuadricópteros también se plantea como líneas de trabajo futuro el uso de visión interna del robot, de modo que sea capaz de ubicarse y encontrar los objetivos completando con mayor precisión su vuelo partiendo de la información que obtiene del ambiente actual.

6 Bibliografía

1. Bitcarze AB, http://www.bitcraze.se/
2. Crazyflie Nano C++ Client Library, https://github.com/fairlight1337/libcflie
3. OpenCV, http://opencv.org/
4. http://wiki.robocup.org/wiki/Small_Size_League
5. Shapiro, L. G. & Stockman, G. C: "Computer Vision". Prentice Hall, 2001
6. Mark S. Nixon & Alberto S. Aguado. Feature Extraction and Image Processing. Academic Press, 2008.
7. J. F. Canny. A computational approach to edge detection. IEEE Trans. Pattern Analysis and Machine Intelligence, 1986.
8. HOUGH, P. V. C. Method and means for recognizing complex patterns. U. S. Patent 3, 069 654, December 18 , 1962.
9. Antolovic, Danko. Review of the Hough Transform Method, With an Implementation of the Fast Hough Variant for Line Detection. Department of Computer Science, Indiana University, & IBM Corporation, 2008.
10. Instituto de Sistemas Inteligentes y Enseñanza Experimental de la Robótica, https://www.facebook.com/isierum