Algoritmo de paralelización para la estimación en tiempo real del ángulo de guiñada de un UAV

Claudio Paz, Gastón Araguás, Gonzalo Perez Paina and Hugo Toloza

Centro de Investigación en Informática para la Ingeniería, Universidad Tecnológica Nacional, Regional Córdoba
Maestro Lopez esq. Cruz Roja Argentina, Córdoba, Argentina
cpaz@scdt.frc.utn.edu.ar

***Abstract*—Determination of the vehicle orientation is fundamental for any autonomous tasks. Magnetometer readings may be inaccurate on indoor vehicles given metallic structures, electric and electronic devices. In this work a real time parallel algorith for yaw determination based on images processing instead of magnetometer readings is presented. The system is implemented in a ground station to aid a hovering unmanned aerial vehicle with an onboard down looking camera. This system is able to perform a real time spectral analysis of the images for rotation determination. Image processing is made on a little Beowulf cluster of desktop machines with shared resources. To assess the accuracy, runtime, scalability and speedup results of the implementation of the system on public data sets are shown.**

***Resumen*—La determinación de la orientación de un vehículo es fundamental para cualquier tarea que se realice autónomamente. En los vehículos que realizan tareas en ambientes interiores las lecturas de los magnetómetros pueden ser erradas debido a estructuras metálicas, dispositivos eléctricos y electrónicos. En este trabajo se propone la implementación de un algoritmo paralelo capaz de determinar en tiempo real el ángulo de guiñada de un vehículo utilizando imágenes en lugar del magnetómetro. El sistema se implementa en una estación terrena asistiendo a un vehículo volador no tripulado en vuelo estacionario con una cámara a bordo apuntando al piso. El sistema es capaz de realizar un análisis espectral en tiempo real de las imágenes tomadas para determinar su rotación. El procesamiento de las imágenes se realiza en un pequeño *cluster* Beowulf formado por máquinas de escritorio con recursos compartidos. Para evaluar la precisión, tiempo de ejecución, escalabilidad y *speedup* se muestran resultados de la aplicación del sistema sobre conjuntos de datos públicos.**

I. Introducción

Los vehículos no tripulados han ganado notoriedad en los últimos años, en particular los vehículos aéreos no tripulados o UAV por las siglas en inglés de *Unmanned Aerial Vehicle*, debido a la posibilidad de utilizarlos en ambientes inaccesibles o peligrosos para seres humanos [1], como así también para minimizar los costos de usar vehículos tripulados en algunas tareas [2]. Además, para las aplicaciones realizadas en ambientes interiores, en los que el espacio es muy reducido son necesarios UAV de tamaño acotado y alta maniobrabilidad. Este es el caso de los cuadricópteros [3], los cuales están formados por una estructura rígida en forma de cruz y cuatro hélices fijas lo que minimiza su costo y les brinda gran agilidad.

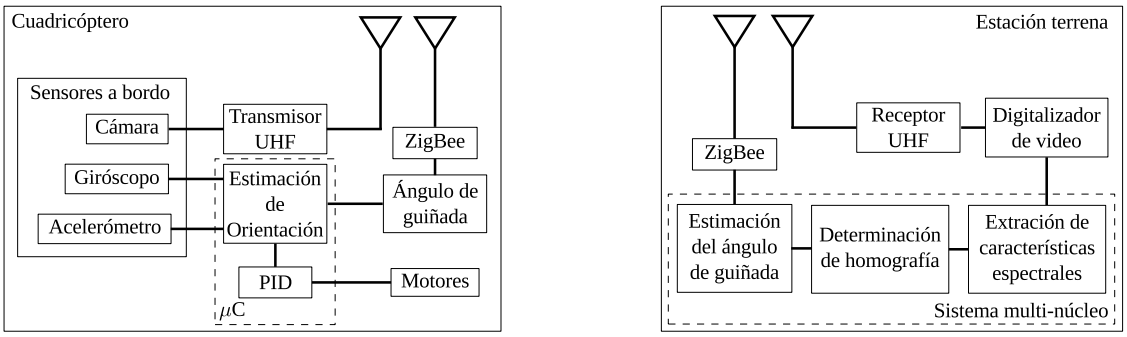
El vuelo de un cuadricóptero se puede dividir en tres etapas ordenadas en creciente dificultad: vuelo estacionario, navegación de un punto a otro y despegue y aterrizaje. Para poder realizar con éxito cualquiera de estas etapas del vuelo se necesita determinar con precisión la posición y la orientación del cuadricóptero para lo cual se cuenta con diversos sensores como acelerómetros, giróscopos, magnetómetros, barómetros, GPS, etc. La orientación se puede representar de distintas formas entre las cuales las más utilizadas para los UAV son los ángulos de Euler, la matriz de cosenos directores (DCM) y los cuaterniones [4]. La representación de orientación por medio de ángulos de Euler es intuitiva en comparación con los cuaterniones y bajo ciertas restricciones puede ser usada en UAV. Se puede definir la orientación de un cuerpo respecto de una referencia fija por medio de tres rotaciones alrededor de sus ejes. Si el eje x positivo apunta hacia adelante, el eje y positivo hacia la derecha y el eje z hacia abajo, entonces la rotación alrededor del eje x se conoce como rolido, alrededor del eje y cabeceo y finalmente, la rotación alrededor del eje z se llama guiñada. Para determinar la orientación de un UAV generalmente se utilizan giróscopos y acelerómetros. En algunos casos se utilizan también magnetómetros, los que proveen de una referencia absoluta del norte magnético. Pero debido a las fluctuaciones de campo magnético en ambientes interiores ocasionadas por estructuras metálicas y dispositivos electrónicos [5], el uso del magnetómetro está aconsejado sólo para exteriores.

Tomando ventaja de que las cámaras tienen bajo peso y bajo costo, y considerando que las imágenes contienen gran cantidad de información, una cámara mirando al piso es un buen reemplazo para el magnetómetro en ambientes interiores. Los cuadricópteros comerciales por lo general cuentan con cámaras entre sus sensores, este es el caso del AR.Drone [6] y el MikroKopter [7].

En vuelo estacionario, una cámara apuntando hacia abajo fija al cuadricóptero tiene el eje focal en posición vertical la mayoría del tiempo. En este caso, con una gran diferencia en velocidad de captura de imágenes comparada con la dinámica del vehículo, se puede aproximar las transformaciones de las sucesivas imágenes como una rotación pura alrededor del eje focal.

Se puede determinar esta rotación entre dos imágenes consecutivas detectando el movimiento de ciertas características en la imagen [8]. Para el caso de una cámara apuntando hacia abajo la mejor opción es la detección de características espectrales utilizando la representación de las imágenes en el dominio de Fourier. Para determinar la rotación de la información espectral encontrada, primero se utiliza el método de correlación de fase para determinar el desplazamiento entre las características asociadas a las imágenes y luego se calcula la transformación que lleva a

978−1−4799−4269−5/14/$31.00



(a) Sistema a bordo del cuadricóptero

(b) Sistema en estación terrena

Fig. 1: Diagrama en bloques del sistema para la determinación de orientación en tiempo real de un cuadricóptero

cabo ese corrimiento para determinar el ángulo de rotación.

El sistema utilizado está compuesto de dos partes: una etapa de bajo nivel, ejecutándose en una placa a bordo del cuadricóptero y otra etapa de alto nivel implementada en una estación terrena con mayores recursos de computo. Dicho sistema se muestra esquemáticamente en la Fig. 1. La etapa de bajo nivel tiene como objetivo mantener el cuadricóptero en un vuelo estable, por lo cual es necesario estimar los ángulos de rolido y cabeceo con gran velocidad. La estimación se realiza mediante la fusión de las lecturas provistas por el acelerómetro y el giróscopo. Además, en esta etapa se controla la potencia de los motores para la estabilización del cuadricóptero mediante un lazo proporcional, integral y derivativo (PID) como se muestra en la Fig. 1a. En la etapa de alto nivel, mostrada en la Fig. 1b, se determina el ángulo de guiñada a partir de las imágenes recibidas desde el cuadricóptero en pleno vuelo. El ángulo estimado se devuelve al cuadricóptero para ser fusionado por la etapa de bajo nivel con la información de los otros sensores.

En este trabajo se propone un esquema de paralelización para determinar el ángulo de guiñada en tiempo real utilizando nodos conectados con una jerarquía de comunicación similar a la configuración Maestro/Esclavo. Este esquema propone una entidad intermedia llamada Capataz que permite disminuir la latencia de las comunicaciones entre nodos.

El trabajo está organizado de la siguiente manera: en la Sec. II se muestran los desarrollos necesarios para fundamentar la obtención del ángulo de guiñada a partir de las imágenes y en la Sec. III se detalla el algoritmo usado en la estación terrena para determinar este ángulo. Finalmente en la Sec. IV se muestran los resultados obtenidos con diversos métodos de división del dominio del problema y en la Sec. V se enumeran las conclusiones del trabajo.

II. Determinación del ángulo de guiñada

Dadas dos imágenes de un mismo plano, obtenidas desde distintos puntos de vista, cada punto característico $\boldsymbol{m}_a$ en la imagen i_a, y su correspondiente $\boldsymbol{m}_b$ en la imagen i_b, están relacionados por una homografía $\boldsymbol{H}_{ba}$ inducida por el plano [8] tal que

$$\boldsymbol{m}_a = \boldsymbol{H}_{ba}\boldsymbol{m}_b \tag{1}$$

Esta homografía representa la traslación y rotación espacial existente entre las dos posiciones de la cámara. En particular, si el movimiento de la cámara se restringe a un plano paralelo al que contiene a las características y si además se desprecian los ángulos de rolido y cabeceo, la homografía $\boldsymbol{H}_{ba}$ queda definida por

$$\boldsymbol{H}_{ba} = \begin{bmatrix} \boldsymbol{R}_z & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix} \tag{2}$$

donde $\boldsymbol{t}$ es el vector que representa la traslación y $\boldsymbol{R}_z$ es la matriz de rotación alrededor del eje z de la cámara. La matriz $\boldsymbol{R}_z$ está definida por

$$\boldsymbol{R}_z = \begin{bmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{bmatrix} \tag{3}$$

siendo ψ el ángulo de rotación entre las dos imágenes.

En la estación terrena, para determinar el ángulo de guiñada primero se deben extraer las características $\boldsymbol{m}_a$ y $\boldsymbol{m}_b$ para luego poder calcular la homografía $\boldsymbol{H}_{ba}$. Finalmente, a partir de la homografía se pude calcular la matriz de rotación $\boldsymbol{R}_z$ y con ella el ángulo de guiñada ψ. En general las características son puntos destacados en las imágenes que representan cambios de intensidad, lo que los hace fácilmente reconocibles en las sucesivas imágenes. En ocasiones, el vuelo de un cuadricóptero se realiza a baja altura o sobre superficies muy regulares, como por ejemplo una alfombra. En estos casos, es difícil encontrar características basadas en intensidad. Se puede usar en cambio la representación de la imagen en el dominio de la frecuencia, obtenida de la transformada de Fourier. Esta representación tiene la ventaja de poder aprovechar el Teorema del corrimiento de Fourier, el cual afirma que la transformada de Fourier de dos imágenes idénticas desplazadas una de la otra en una cantidad (u, v), sólo difieren en fase. Es decir, dadas las imágenes

$$i_a(x, y) = i_b(x + u, y + v) \tag{4}$$

sus transformadas de Fourier están relacionadas por

$$I_a(\omega_x, \omega_y) = e^{j(u\omega_x + v\omega_y)} I_b(\omega_x, \omega_y) \tag{5}$$

Este desplazamiento de fase se puede encontrar usando el espectro cruzado de potencia o CPS (*Cross-Power Spectrum*) de la siguiente forma

$$\mathcal{C}(I_a, I_b) = \frac{I_a(\omega_x, \omega_y) I_b^*(\omega_x, \omega_y)}{|I_a(\omega_x, \omega_y)||I_b^*(\omega_x, \omega_y)|} = e^{j(u\omega_x + v\omega_y)} \tag{6}$$

donde I_a e I_b son las transformadas de Fourier de i_a e i_b respectivamente y donde el símbolo $*$ representa la operación de conjugación en el espacio de los números complejos. Finalmente, para recuperar el desplazamiento

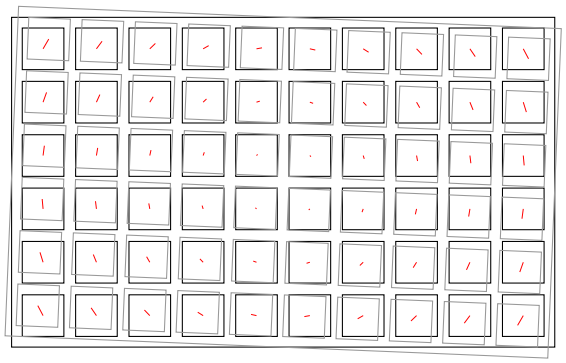


Fig. 2: Desplazamiento de los parches entre dos imágenes

(u, v) se debe operar con la transformada inversa de Fourier. En la práctica, el algoritmo usado es la transformada rápida de Fourier o FFT (*Fast Fourier Transform*) y la transformada inversa (IFFT) para operar luego del CPS.

Debido a que para poder recuperar una homografía se necesitan al menos cuatro puntos correspondientes, se pueden dividir las imágenes en varias partes o parches p_i y obtener la Transformada de Fourier de cada una de ellas. De esta forma se obtiene un desplazamiento $\Delta \boldsymbol{d}_i$ por cada uno de los parches utilizados como se muestra en la Fig. 2. Sumando estos desplazamientos a algún punto de los parches de la primera imagen (por ejemplo su centro) se obtiene el conjunto de características correspondientes entre la primera y la segunda imagen

$$\{\boldsymbol{m}_{a_i} \leftrightarrow \boldsymbol{m}_{a_i} + \Delta \boldsymbol{d}_i = \boldsymbol{m}_{b_i}\} \tag{7}$$

En el caso de contar con más de cuatro características hay que determinar si existen correspondencias mal calculadas y eliminarlas del conjunto. Para esto se utiliza un algoritmo iterativo como RANSAC [9]. De esta forma se puede aumentar la precisión en la determinación del ángulo.

Luego, encontrada la homografía que relaciona las características de la primera imagen con las características correspondientes en la segunda imagen se debe extraer la matriz de rotación y finalmente el ángulo de guiñada ψ [10]. En el Alg. 1 se muestra de forma simplificada la función para realizar la estimación del ángulo mientras que en el Alg. 2 se calcula el desplazamiento usando las características espectrales.

III. Implementación en arquitecturas multi-nodo

En este trabajo se propone implementar el algoritmo descripto en la Sec. II en una arquitectura paralela utilizando un paradigma de descomposición de dominio que aproveche la división en parches de la imagen. Para maximizar la cantidad de procesadores o núcleos entre los cuales dividir los parches se propone utilizar un arreglo de computadoras de escritorio formando un pequeño *cluster* Beowulf [11]. Una jerarquía Maestro-Esclavo se vería afectada en este caso por la latencia de la red debido a la división y el envío de los pequeños bloques a cada procesador. Para minimizar el tiempo de comunicación entre los procesadores se propone una jerarquía diferente a la tradicional Maestro-Esclavo. Los esclavos son agrupados en conjuntos de procesadores en función de su ubicación física y controlados por un procesador de menor rango que el maestro llamado capataz, de esta forma, el maestro se comunica con los distintos capataces y estos con sus esclavos. Esta jerarquía es especialmente útil si los esclavos de cada capataz comparten memoria, de modo que la comunicación se realiza internamente minimizando la latencia. Este es el caso de los procesadores que se encuentran en el mismo nodo. La descomposición de la imagen en los parches y la distribución entre los capataces y los esclavos está esquematizada en la Fig. 3. Si bien con este esquema de descomposición de los datos el mensaje a enviar a los capataces es de mayor tamaño, la cantidad de comunicaciones necesarias son menores con lo que se logra ocultar la latencia de la red. Luego de que cada esclavo calcule los desplazamientos $\Delta \boldsymbol{d}_i$ que le corresponden se los

Algoritmo 1 Estimación del ángulo de guiñada

function ESTIMACION_GUIÑADA(i_t, i_{t-1})
  Obtener parches $p_{i\,t}$ y $p_{i\,t-1}$ a partir de i_t y i_{t-1}
  for all $\{p_{i\,t}, p_{i\,t-1}\}$ **do**
    $\Delta \boldsymbol{d}_i \leftarrow$ DESPLAZAMIENTO($p_{i\,t}, p_{i\,t-1}$)
    $\boldsymbol{m}_{i\,t} \leftarrow \boldsymbol{m}_{i\,t-1} + \Delta \boldsymbol{d}_i$
  end for
  $\psi \leftarrow \boldsymbol{R}_z \leftarrow \boldsymbol{H} \leftarrow$ HOMOGRAFÍA($\boldsymbol{m}_{i\,t}, \boldsymbol{m}_{i\,t-1}$)
  return ψ
end function

Algoritmo 2 Determinación del desplazamiento entre parches

function DESPLAZAMIENTO($p_{i\,t}, p_{i\,t-1}$)
  $P_{i\,t} \leftarrow FFT(p_{i\,t})$
  $P_{i\,t-1} \leftarrow FFT(p_{i\,t-1})$
  $R \leftarrow CPS(P_{i\,t}, P_{i\,t-1})$
  $r \leftarrow IFFT(R)$
  $\Delta \boldsymbol{d}_i \leftarrow$ MAX(r)
  return $\Delta \boldsymbol{d}_i$
end function

Algoritmo 3 Implementación paralela de Alg. 1

function ESTIMACION_GUIÑADA_PARALELA(i_t, i_{t-1})
  if Procesador == Maestro **then**
    Obtener bloques $b_{j\,t}$ y $b_{j\,t-1}$ a partir de I_t y I_{t-1}
    Enviar bloques a los j capataces
  else if Procesador == Capataz **then**
    Recibir bloques del maestro
    Obtener parches $p_{i\,t}$ y $p_{i\,t-1}$ a partir de $b_{j\,t}$ y $b_{j\,t-1}$
    Enviar parches a los i esclavos
  else if Procesador == Esclavo **then**
    $\Delta \boldsymbol{d}_i \leftarrow$ DESPLAZAMIENTO($p_{i\,t}, p_{i\,t-1}$)
  end if
  GATHER($\Delta \boldsymbol{d}_i$)
  if Procesador == Maestro **then**
    $\boldsymbol{m}_{i\,t} \leftarrow \boldsymbol{m}_{i\,t-1} + \Delta \boldsymbol{d}_i$
    $\psi \leftarrow \boldsymbol{R}_z \leftarrow \boldsymbol{H} \leftarrow$ HOMOGRAFÍA($\boldsymbol{m}_{i\,t}, \boldsymbol{m}_{i\,t-1}$)
    return ψ
  end if
end function

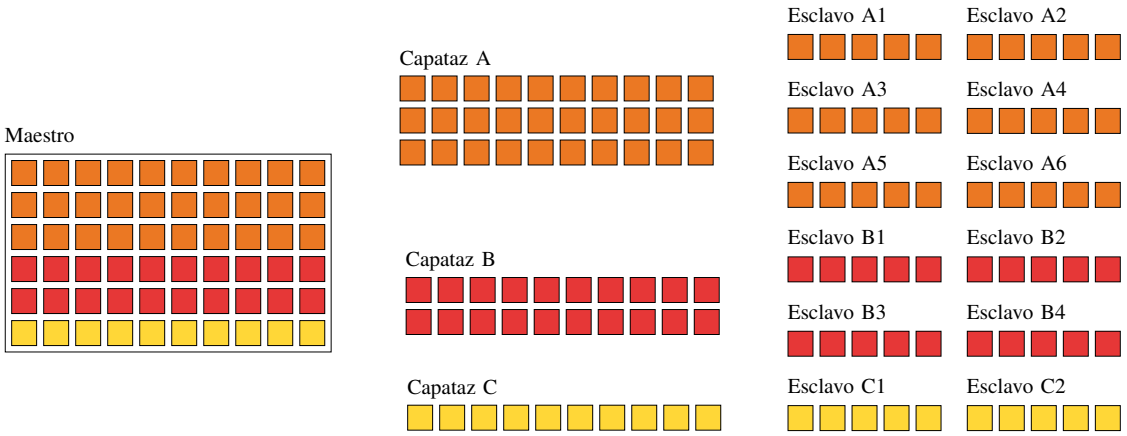


Fig. 3: Partición del dominio entre los capataces antes del envío a los esclavos para optimizar las comunicaciones

envía a su capataz. Finalmente, las distancias reunidas por los capataces son enviadas al maestro para que éste calcule la homografía y consecuentemente el ángulo de guiñada ψ. El Algoritmo 3 resume el método de partición de dominio propuesta.

El *cluster* Beowulf en donde funciona la estación terrena está formado por dos nodos con procesadores Intel i5 de 4 núcleos y un nodo de 16 núcleos (compuesto por 2 procesadores AMD Opteron de 8 núcleos cada uno) conectados al mismo segmento de una red LAN. Si bien este último nodo es más antiguo y su poder de computo es similar a los otros nodos, se lo consideró para la estación terrena debido a que su gran cantidad de núcleos permite una mejor evaluación de la escalabilidad del algoritmo paralelo.

Debido a que el *cluster* resultante es un sistema no homogéneo se utiliza un método de balance de carga simple basado en el porcentaje de uso de cada unidad de procesamiento. El método de balance consiste en disminuir la cantidad de parches enviados a los núcleos más lentos y aumentar la cantidad enviada a los más rápidos basándose en una variable de peso. Se utiliza como variable de peso al tiempo que le llevaba a cada núcleo realizar el cálculo solicitado.

IV. Resultados

Para evaluar el sistema se utilizó un conjunto de datos públicos [12] compuesto de imágenes tomadas a bordo de un cuadricóptero en un vuelo estacionario o *cuasi* estacionario de aproximadamente 100 segundos. Las imágenes son de 752×480 píxeles tomadas por una cámara apuntando hacia abajo fija al chasis del cuadricóptero a una velocidad de captura de 20 cuadros por segundo. El conjunto de datos también provee la posición y orientación relativa a un marco local dadas por un sistema de seguimiento externo además de las lecturas de una unidad inercial a bordo.

Para la implementación del Alg. 2 se utilizó la biblioteca OpenCV que cuenta con la función `phaseCorrelate()` la cual recibe como argumentos dos imágenes y devuelve el desplazamiento (u, v) entre ellas utilizando el método descripto en la Sec. II. Al usar esta función, en cada paso se vuelve a calcular la FFT de la imagen provista en el paso anterior, lo cual es ineficiente. Sin embargo, tanto el algoritmo secuencial como el paralelo utilizan la misma función, por lo que la comparación es justa.

Se evaluaron distintas configuraciones utilizando el modelo de comunicación tradicional Maestro/Esclavo y el modelo propuesto Maestro/Capataz/Esclavo, ambos con y sin balance de carga. También se pusieron a prueba distintas cantidades de nodos y núcleo utilizando parcialmente el nodo con 16 núcleos. En todos los casos, el proceso maestro así como también los capataces realizaron las tareas de los esclavos. Para identificar en las gráficas al número de nodos y núcleos se utilizan las letras H y N mayúsculas respectivamente, precedidas de un número indicando la cantidad de nodos y núcleos.

La evaluación de velocidad se llevó a cabo considerando el tiempo que le insume al algoritmo realizar la estimación en cada ciclo, incluyendo la carga de la nueva imagen en memoria desde el disco, hasta que el ángulo es estimado. Para las gráficas se utilizó el valor medio de cada parámetro medido sobre cada ciclo de 2041 imágenes del conjunto de datos en 10 iteraciones.

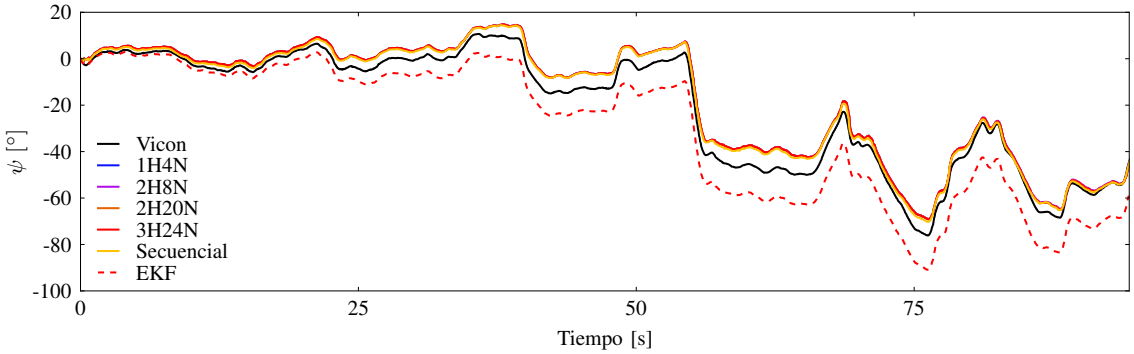
Para contrastar la precisión en la estimación del ángulo de guiñada se utilizó la medición provista por un sistema de seguimiento Vicon la cual es considerada para este trabajo como valor verdadero.

La determinación cuantitativa de la precisión del algoritmo de estimación fue realizada mediante el error medio acumulado [13]. El error medio acumulado ε_N^k para la $k-$ésima muestra de N imágenes consecutivas está dado por

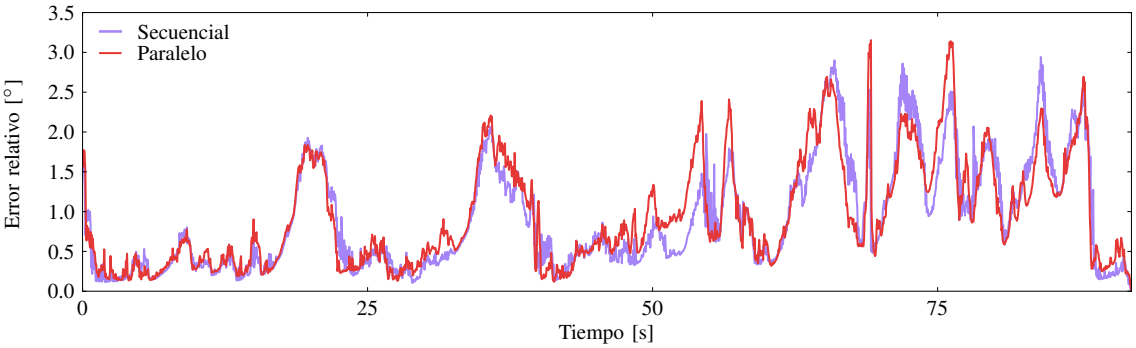
$$\varepsilon_N^k = \frac{1}{N} \sum_{i=k}^{k+N} \left| \psi_e^i - \psi_e^k \right|, \quad k = 1, \ldots, M - N \qquad (8)$$

donde M es el número total de muestras.

La Fig.4a muestra los ángulos obtenidos por las distintas configuraciones implementadas en el *hardware* disponible, así como también el provisto por el sistema Vicon. Como se ve, las curvas correspondientes a las implementaciones paralelas son coincidentes, lo que indica que la forma de dividir los parches entre los nodos no afecta a la estimación. Además, se observa que para las distintas cantidades de nodos y núcleos se logra obtener de manera correcta el ángulo de guiñada. Ésto permite asegurar que el proceso de paralelización no modifica la precisión del algoritmo. También en esta figura se muestra el resultado de la estimación del ángulo de guiñada realizado por un Filtro Extendido



(a) Ángulo de guiñada estimado por las aplicaciones paralelas básicas y la secuencial original. También se muestra la estimación dada por la unidad inercial sin imágenes utilizando EKF y el valor verdadero dado por el sistema de seguimiento Vicon.



(b) Error relativo en la estimación utilizando el error medio acumulado dado por (8) para un $N = 100$. La aplicación original se usa para el resultado secuencial mientras que para resultado paralelo se utiliza una configuración con 1 maestro y 24 esclavos.

Fig. 4: Estimación del ángulo de guiñada.

de Kalman (EKF por sus siglas en Ingles) utilizando sólo la integración de los giróscopos, es decir, sin realizar la corrección a partir de los datos provistos por la estación terrena.

Se observa que el EKF implementado sólo con giróscopos tiene divergencia en el ángulo de guiñada lo que justifica el uso de imágenes para corrección de dicho ángulo. El error medio acumulado dado por (8) para un $N = 100$ se muestra en la Fig. 4b donde su valor máximo es de aproximadamente $3°$, y está muy por debajo de este valor la mayor parte del tiempo.

En la Fig. 5 se muestran los tiempos de ejecución de todos los métodos y configuraciones evaluadas incluyendo el algoritmo secuencial original y el algoritmo paralelo corriendo en un solo núcleo. Tanto el algoritmo secuencial como el paralelo corriendo en un solo núcleo fueron ejecutado en el nodo que se consideró el principal en las demás corridas. En esta figura se muestra que las configuraciones con balance de carga superan en desempeño a casi todas las demás. No así en el caso de las configuraciones homogéneas, como es el ejemplo de la configuración 2H8N en donde se tiene el mismo desempeño siempre que se usen capataces. También se puede apreciar que el balance de carga es fundamental cuando los nodos no son homogéneos.

El mayor incremento de velocidad en la paralelización se logra con el modelo de capataces propuesto en este trabajo, como es el caso de las siete configuraciones más veloces. Estas siete configuraciones son las únicas capaces de asegurar un tiempo menor a 0.10s en la estimación del ángulo de guiñada. Esto significa que en el *hardware* propuesto, usando capataces y balance de carga, la estimación del ángulo de guiñada se puede realizar a una frecuencia de $10Hz$, suficientes para cerrar el lazo de control correspondiente en tiempo real. Sin embargo, es importante destacar que el uso del modelo con capataces mejora los

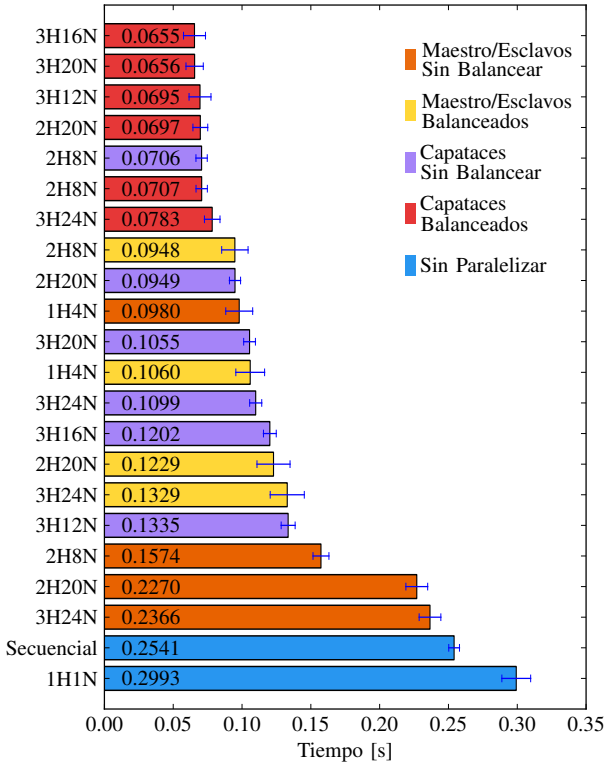


Fig. 5: Tiempos de ejecución de cada configuración implementada.

tiempos de ejecución en todas las configuraciones utilizadas comparado con el tradicional modelo Maestro/Esclavo.

Una métrica muy utilizada en computación paralela es el *speedup* algorítmico s_N, que relaciona los tiempos de ejecución de cada configuración con la aplicación secuencial original dado por

$$s_N = \frac{t_1}{t_N} \tag{9}$$

donde t_1 es el tiempo de ejecución de la aplicación original y t_N es el tiempo de ejecución de cada configuración paralela implementada, siendo N la cantidad de procesos utilizados.

Se puede obtener el *speedup* de cada implementación teniendo en cuenta los tiempos mostrados en la Fig. 5. Aquí se evidencia la falta de escalabilidad del algoritmo siendo los casos más representativos las configuraciones más veloces 3H16N y 3H20N. La configuración 3H16N está formada por el nodo principal, otro nodo de iguales características, ambos con 4 núcleos y un nodo extra con 16 núcleos de los cuales sólo se ocupan 8. Por otro lado la configuración 3H20N tiene los mismos nodos pero ocupando 12 núcleos de los 16 disponibles en el nodo extra. Ambas configuraciones implementadas con capataces y balance de carga. Se deduce de la figura que en estas condiciones agregar 4 núcleos de iguales características no representa ninguna mejora en el desempeño de la aplicación.

Otro caso semejante se presenta en las dos mejores configuraciones 2H8N (capataces balanceados y sin balancear) con el agregado de otro nodo de 4 núcleos (de 16 disponibles) para formar la configuración 3H12N. En el caso de la versión balanceada, agregar el nodo extra no presenta mejoras en la velocidad de procesamiento que puedan ser consideradas. Sin embargo, debido a que el nodo agregado es de menor capacidad de computo que los que forman el 2H8N, en la versión sin balancear la configuración 3H12N empeora el desempeño de manera considerable, llevando el tiempo de ejecución a casi el doble.

Finalmente, como caso crítico, el *speedup* de las configuraciones Maestro/Esclavo 3H24N y 2H20N sin balance de carga usando (9) da un valor de 1.07 y 1.12 respectivamente lo que no representa mejora significativa respecto de la aplicación serial. Sin embargo, con capataces y balance de carga las mismas configuraciones obtienen un *speedup* de 3.24 y 3.65 respectivamente, logrando en ambos casos estar por debajo de los 0.10s en el tiempo de ejecución.

V. Conclusiones

En este trabajo se propuso un esquema de paralelización en una estación terrena para estimar el ángulo de guiñada de un UAV en tiempo real. La estación terrena consiste en un *cluster* Beowulf compuesto de equipos de oficina con un equipo principal que recibe las imágenes, las fragmenta y envía los parches a los otros equipos para procesarlos. e utilizó un método de descomposición de dominio en el que cada parche es procesado por un núcleo con un balance de carga en donde los núcleos más rápidos reciben más parches que los lentos dependiendo del tiempo de proceso de cada parche.

Se propuso un modelo de comunicación Maestro/Capataz/Esclavo que consiste en el envío de grandes bloques de datos a procesos llamados capataces para que luego estos envíen los porciones menores a los procesos esclavos con los que comparte memoria, ocultando así la latencia de la red. Con este modelo se obtuvo un *speedup* mucho mayor al obtenido con el tradicional Maestro/Esclavo. Se pusieron a prueba distintas configuraciones utilizando 3 nodos con un total de 24 núcleos. Los resultados mostraron que el algoritmo puede ser utilizado para cerrar un lazo de control de ángulo de guiñada en tiempo real.

En cuanto a la escalabilidad, debido a la división de dominio elegida, el sistema sólo puede escalar hasta una cantidad de procesos igual al número de parches de la imagen.

Como trabajo futuro se plantea la implementación propia de la función `phaseCorrelate()` de modo que no recalcule la FFT obtenida en el paso anterior, de lo que se espera una gran mejora en el *speedup*. Además, se propone la implementación en GPU, ya que el mayor cuello de botella se observó en las comunicaciones.

Agradecimientos

El primer autor se financia con el programa de becas de la Universidad Tecnológica Nacional. Este trabajo se enmarca dentro del proyecto "Guiado de Vehículos Autónomos usando Fusión de Señales de GPS de Bajo Costo y otros Sensores", PICT-PRH-2009-0136.

Referencias

[1] J. Neto, R. da Paixao, L. Rodrigues, E. Moreira, J. dos Santos, and P. Rosa, "A surveillance task for a uav in a natural disaster scenario," in *Industrial Electronics (ISIE), 2012 IEEE International Symposium on*, 2012, pp. 1516–1522.

[2] T. Moranduzzo and F. Melgani, "Automatic car counting method for unmanned aerial vehicle images," vol. 52, no. 3, pp. 1635–1647, 2014.

[3] S. Gupte, P. Mohandas, and J. Conrad, "A survey of quadrotor unmanned aerial vehicles," in *Southeastcon, 2012 Proceedings of IEEE*, 2012, pp. 1–6.

[4] W. Phillips and C. Hailey, "Review of attitude representations used for aircraft kinematics," *Journal of Aircraft*, vol. 38, no. 4, pp. 718–737, 2001.

[5] K. Yamazaki, K. Kato, K. Ono, H. Saegusa, K. Tokunaga, Y. Iida, S. Yamamoto, K. Ashiho, K. Fujiwara, and N. Takahashi, "Analysis of magnetic disturbance due to buildings," *Magnetics, IEEE Transactions on*, vol. 39, no. 5, pp. 3226–3228, 2003.

[6] *The Navigation and Control Technology Inside the AR.Drone Micro UAV*, Milano, Italy, 2011.

[7] I. Sa and P. Corke, "System identification, estimation and control for a cost effective open-source quadcopter," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, May 2012, pp. 2202–2209.

[8] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.

[9] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[10] G. Araguas, C. Paz, D. Gaydou, and G. P. Paina, "Orientation estimation fusing a downward looking camera and inertial sensors for a hovering UAV," in *Advanced Robotics (ICAR), 2013 16th International Conference on*, Nov 2013, pp. 1–6.

[11] T. Sterling, D. Becker, D. Savarese, J. Dorband, U. Ranawake, and C. Packer, "Beowulf: A parallel workstation for scientific computation," in *In Proceedings of the 24th International Conference on Parallel Processing*. CRC Press, 1995, pp. 11–14.

[12] G. H. Lee, M. Achtelik, F. Fraundorfer, M. Pollefeys, and R. Siegwart, "A benchmarking tool for mav visual pose estimation," in *Control Automation Robotics & Vision (ICARCV), 2010 11th International Conference on*, 2010, pp. 1541–1546.

[13] N. Nourani-Vatani, J. Roberts, and M. Srinivasan, "Practical visual odometry for car-like vehicles," in *IEEE International Conference on Robotics and Automation, 2009. ICRA '09*, May 2009, pp. 3551 –3557.