Un sistema de visión global paralelo para fútbol de robots con fines educativos

Rodrigo Cañibano, Eduardo Grosclaude, Javier Balladini

Universidad Nacional del Comahue
Buenos Aires 1400, Neuquén Capital, Argentina
{rcanibano, eduardo.grosclaude, javier.balladini}@fi.uncoma.edu.ar

Resumen Se presenta un nuevo sistema de visión global por computadora para el fútbol de robots de la *Small Size League*, que puede ser utilizado como herramienta educativa en una asignatura de visión por Computadora y permite explorar distintas estrategias de paralelización. El sistema procesa cuadros de vídeo y reporta la posición y orientación de los robots y la posición de la pelota en cada uno de ellos. Se aplicaron conjuntamente dos estrategias de paralelización. Una de ellas explota el paralelismo dentro de cada cuadro, dividiendo los cuadros en fragmentos que son procesados de forma independiente. La otra estrategia se basa en el procesamiento simultáneo de diferentes cuadros del vídeo. Se realizó una implementación en OpenMP para C++, basada en plugins para facilitar su modificación en un ambiente educativo. En un servidor con un procesador Intel Xeon E5-2630 (6 cores y multithreading simultáneo), la solución paralela respecto a la solución que utiliza un único núcleo (de los 6 disponibles) logró mejoras máximas de 5,27x en los FPS procesados mientras el retardo por cuadro se redujo a la mitad.

1. Introducción

La RoboCup[2] (del Inglés *Robot World Cup*) es una competencia internacional celebrada desde 1997, donde equipos de robots juegan una versión simplificada del fútbol. Su finalidad es la de poner a prueba los avances en distintas áreas de conocimiento como la inteligencia artificial, visión por computadora y robótica. En la la liga de robots de tamaño pequeño denominada *Small Size League* (SSL) de la RoboCup los robots tienen capacidad de procesamiento reducida, delegando la toma de decisiones a la computadora de su equipo, que percibe el ambiente a través de un sistema de visión global centralizado compartido. Se utiliza un conjunto de cámaras montadas sobre distintas áreas del campo de juego, conectadas a una computadora donde se ejecuta el sistema de visión. El sistema detecta la posición y orientación de cada uno de los robots (en base a parches de colores) y la posición de la pelota, y reporta esta información a las computadoras que controlan los equipos.

Los criterios de calidad de este tipo de sistema son, normalmente: tasa de aciertos en la detección de objetos, precisión en la posición y orientación de los robots y pelota, cuadros por segundo (FPS, del inglés *Frames Per Second*) y el

retardo de cuadro (el tiempo desde la creación del cuadro hasta la entrega de la información extraída) máximo.

Desde sus orígenes, el tamaño de la cancha se ha ido incrementando. En el 2015 se establecieron como predeterminadas las canchas de tamaño doble para las cuales son necesarias cuatro cámaras en lugar de las dos utilizadas por las canchas de tamaño simple[1]. Duplicar la cantidad de información que se debe procesar impacta negativamente en el rendimiento de estos sistemas, debido a que se reducen los FPS procesados y se incrementa el retardo de cuadro. Para aumentar el rendimiento de estos sistemas es necesario implementar soluciones paralelas que aprovechen las múltiples unidades de procesamiento de las arquitecturas de cómputo actuales.

Actualmente el software de visión global utilizado en la SSL es el *SSL-Vision*[4]. Éste es un sistema basado en plugins, lo que le permite ser extendido fácilmente. Lamentablemente, presenta algunas desventajas en relación a la escalabilidad y al uso educativo. El sistema de captura de cuadros y el de procesamiento están fuertemente acoplados, formando parte del mismo hilo de ejecución. Esto impacta tanto en el rendimiento (sólo utiliza un núcleo de múltiples disponibles) como en la dificultad de utilizar el sistema como herramienta educativa en temas de paralelismo.

En [3] se presenta un framework destinado al uso educativo, basado en plugins organizados en pilas. El framework tiene dos hilos de búsqueda de objetos, uno para encontrar la pelota y otro para encontrar los robots, y un hilo de captura de cuadros de vídeo. Su grado de paralelismo es mejor que el de *SSL-Vision* pero aún así no permite escalar a más de tres núcleos.

El objetivo de este trabajo es el de crear un nuevo sistema de visión global por computadora para el fútbol de robots de la SSL, que pueda ser utilizado como herramienta educativa en una asignatura de visión por Computadora y permita explorar distintas estrategias de paralelización.

El resto de este trabajo está organizado de la siguiente manera. En la sección 2 se describe el sistema propuesto detallando las estrategias que se aplican, las tareas del sistema y cómo se dividen los cuadros. En la sección 3 se describen los resultados experimentales. La sección comienza explicando la metodología y plataforma experimentales; luego se presentan y analizan los resultados de los experimentos. Finalmente, en la sección 4, se discute el cumplimiento de los objetivos propuestos y se plantean líneas de trabajo futuras.

2. Descripción del sistema propuesto

El sistema debe procesar cuadros de un vídeo y reportar la posición y orientación de los robots y la posición de la pelota en cada uno de ellos. Este problema permite aplicar dos estrategias de paralelización:

- Dado que en un vídeo ya decodificado el procesamiento de cada cuadro es independiente del de los demás, se pueden procesar múltiples cuadros al mismo tiempo.

- Para realizar la búsqueda de los objetos, el cuadro puede ser dividido en varias partes que pueden ser procesadas en paralelo.

El sistema ejecuta tareas estáticas y tareas dinámicas. Las tareas estáticas son aquellas que permanecen en ejecución desde el inicio del programa hasta su finalización. Las tareas dinámicas son creadas para procesar un cuadro o fragmento específico y una vez que terminan de procesar el cuadro o fragmento finalizan.

Según su funcionalidad, las tareas son clasificadas en cuatro tipos:

Generación de cuadros: Es la tarea encargada de generar o capturar cuadros y colocarlos en la cola de cuadros a procesar. Ésta es una tarea estática, y sólo hay una en el sistema.

Generación de tareas de fragmentación de cuadro: Esta tarea crea una tarea de fragmentación de cuadro para cada cuadro en la cola. Ésta es una tarea estática, y solo hay una en el sistema.

Fragmentación de cuadro: Ésta es una tarea dinámica. Cada tarea de este tipo divide su cuadro en una cantidad predefinida de fragmentos de igual (o similar) tamaño. Luego, por cada fragmento, se crea una tarea de procesamiento de fragmento.

Procesamiento de fragmento: Ésta es una tarea dinámica. Cada una de estas tareas toma su fragmento asociado y lo procesa utilizando cada una de las pilas de plugins.

Cada una de las dos tareas estáticas tiene un hilo de ejecución asignado. Las tareas dinámicas son ejecutadas por un conjunto de N hilos de ejecución (N es un parámetro de entrada del programa). Cuando un hilo de ejecución (del conjunto) está libre, toma una nueva tarea dinámica para ejecutar. En la figura 1 se muestra la manera en que las tareas son asignadas a los hilos de ejecución.

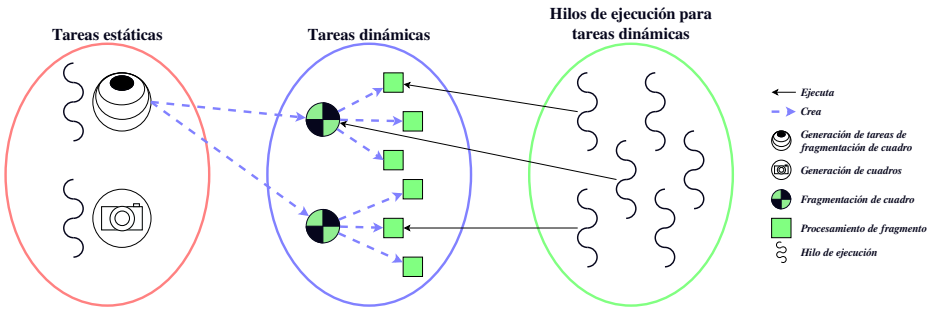

Figura 1. Asignación de tareas a los hilos de ejecución.

Si el cuadro se divide en partes sin zonas solapadas puede suceder que los parches de un robot se encuentren en fragmentos distintos. Para asegurar la detección de todos los parches, cada par de fragmentos adyacentes debe compartir una zona al menos igual al diámetro de un robot (Fig. 2).

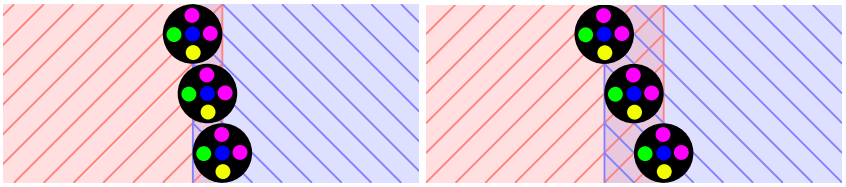

Figura 2. Izquierda: si el área compartida es demasiado pequeña y el robot se encuentra entre dos fragmentos, puede que sus parches no sean completamente visibles desde ninguno de ellos. Derecha: si el área compartida es del ancho de los robots, entonces todos los parches son visibles completamente desde por lo menos un fragmento.

El área compartida mínima resulta minimizando el perímetro de los fragmentos. Esta condición se logra cuando la relación entre su ancho y alto es lo más cercana posible a 1. Dado que el retardo de cuadro es igual al mayor tiempo de retardo de sus fragmentos, es importante balancear la carga. Por esto, todos los fragmentos serán de igual tamaño a excepción de aquellos que se encuentren en los bordes inferior y derecho (que podrían ser más pequeños).

En la figura 3 se muestra cómo se divide una imagen de 800x600 píxeles con un área compartida de 50 píxeles (igual al ancho del robot) en 1, 2, 5 y 8 fragmentos. Cuando el número de fragmentos es un número sin divisores, como en el caso de 5 fragmentos, el cuadro se divide en franjas de igual ancho o altura que el cuadro original. Este tipo de división produce fragmentos con perímetros mayores, produciendo grandes áreas compartidas.

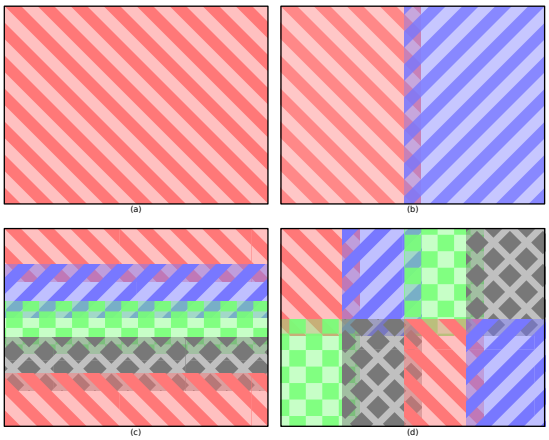

Figura 3. Resultado de dividir un cuadro de 800x600 píxeles con un área compartida de 50 píxeles en 1(a), 2(b), 5(c) y 8(d) fragmentos.

La implementación del sistema se realizó en OpenMP para C++ y está basada en plugins para facilitar su modificación en un ambiente educativo. Se reutilizaron los plugins implementados en [3], modificándolos para permitir su uso en un sistema paralelo.

3. Resultados experimentales

3.1. Metodología experimental

Se configuraron diferentes experimentos haciendo variar la cantidad de partes en las que fueron divididos los cuadros entre 1 y 24, la cantidad de hilos de ejecución para tareas dinámicas entre 1 y 12, y las resoluciones del vídeo entre 800x600 píxeles y 1280x720 píxeles. Para comparar los resultados de las diferentes configuraciones se midieron los FPS soportados por el sistema y el tiempo de retardo máximo (obtenido a los FPS soportados por el sistema bajo la misma configuración).

Como la obtención del retardo máximo depende del conocimiento previo de los FPS, cada experimento se dividió en dos partes. En la primera parte se busca la cantidad de FPS soportados por el sistema para esa configuración. Para esto, el programa procesa tantos cuadros del vídeo como pueda en un cierto intervalo de tiempo. En la segunda parte se limita la cantidad de cuadros por segundo (según el valor de FPS encontrado en la primera parte del experimento) y se registra el tiempo de retardo máximo. Dado que se trata de un sistema de tiempo real, es importante determinar las cotas inferiores del rendimiento. Por esto, se repitió cada experimento 10 veces, registrando sólo el peor valor obtenido para cada variable en cada configuración.

Dada la gran cantidad de experimentos, se buscó el tiempo mínimo de ejecución necesario para que éstos fueran representativos de una ejecución prolongada. Se realizaron tres ejecuciones de 6 minutos cada una con una misma configuración: 11 hilos y 10 fragmentos procesando el vídeo de 800x600 píxeles de resolución. El primer minuto no se contabilizó, con el fin de permitir que la ejecución se estabilizara. Los resultados se compararon con ejecuciones bajo la misma configuración, pero ejecutando de 11 a 20 segundos, y contabilizando sólo los últimos 10 segundos. En la tabla 1 se muestran los FPS obtenidos para cada ejecución. Se observa que las pruebas de 16 segundos son representativas de ejecuciones más largas. Dado que los 6 minutos de vídeo no caben en la memoria RAM del equipo de pruebas, el sistema fue modificado para reintroducir en la lista de cuadros a procesar aquellos cuadros ya procesados. Debido a esto, el rendimiento en estos experimentos es levemente inferior al resto con igual configuración.

3.2. Plataforma experimental

El equipo de pruebas cuenta un procesador Intel Xeon E5-2630. Este es un procesador de 6 núcleos con multithreading simultáneo de dos vías, una frecuencia básica de CPU de 2,3GHz y 2,8GHz en modo turbo, 15MiB de memoria

Duración	6’	11"	12"	13"	14"	15"	16"	17"	18"	19"	20"
Ejecución 1	166	171	170	170	170	169	166	166	166	166	165
Ejecución 2	166	171	170	170	170	168	167	166	166	165	166
Ejecución 3	166	171	170	170	170	169	166	166	167	166	165

Tabla 1. Búsqueda de duración mínima de los experimentos. Duración en segundos(") y minutos (’).

caché L3 compartida entre todos los núcleos, y por cada núcleo hay 256KiB de caché L2 unificada, 32KiB de caché L1 para datos, y 32 KiB de caché L1 para instrucciones. El equipo posee 16GiB de memoria RAM.

3.3. Resultados

En la figura 4 se muestra la cantidades de FPS alcanzados para un vídeo de 800x600 píxeles y 1280x720 píxeles, para distintas cantidades de hilos de ejecución para tareas dinámicas y fragmentos. Note que el sistema ejecuta dos hilos más para las tareas estáticas, uno para la generación de cuadros y otro para la generación de tareas de fractura de cuadro. En el caso del vídeo de 800x600 píxeles se obtuvo un máximo de 195 FPS cuando se divide el cuadro en 15 fragmentos y se utilizan 10 u 11 hilos de ejecución para tareas dinámicas, mientras que para el vídeo de 1280x720 píxeles se alcanzó un máximo de 94 FPS cuando se divide el cuadro en 21 fragmentos y se utilizan 10 u 11 hilos de ejecución para tareas dinámicas. Con respecto a la solución que utiliza un solo hilo de ejecución para tareas dinámicas y un único fragmento, se obtuvo una mejora de 5,27x para el vídeo de 800x600 píxeles y una mejora de 11,75x para el vídeo de 1280x720 píxeles.

En la figura 4 se pueden observar dos patrones que ocurren en ambos vídeos. El primero es que si los cuadros se dividen en 7, 11, 17, 19 y 23 fragmentos, se produce una notable reducción en la cantidad de FPS con respecto a las divisiones adyacentes. Esto se produce porque los cuadros se dividieron en un número primo de fragmentos, y entonces el área total a procesar se incrementó de forma significativa con respecto a los valores circundantes. En la figura 5 se muestra la fluctuación del área para cada cantidad de fragmentos y los FPS obtenidos en divisiones de 1 hasta 100 fragmentos para un vídeo de 1280x720 píxeles de resolución, resaltando los resultados para valores primos y para los valores inmediatamente inferiores y superiores. Como puede observarse, a partir de 7 fragmentos el área total se incrementa de forma significativa cuando el número es primo con respecto a los valores cercanos con divisores. En estos casos, un pequeño incremento del número de fragmentos aumenta el paralelismo, pero los FPS se reducen debido a un gran aumento en el área a procesar.

El segundo patrón que se puede observar en la figura 4 es que cuando la cantidad de fragmentos es baja y la cantidad de hilos de ejecución para tareas dinámicas es alta, el sistema tiene menor FPS que utilizar una menor cantidad de hilos de ejecución para tareas dinámicas pero una mayor cantidad de fragmentos.

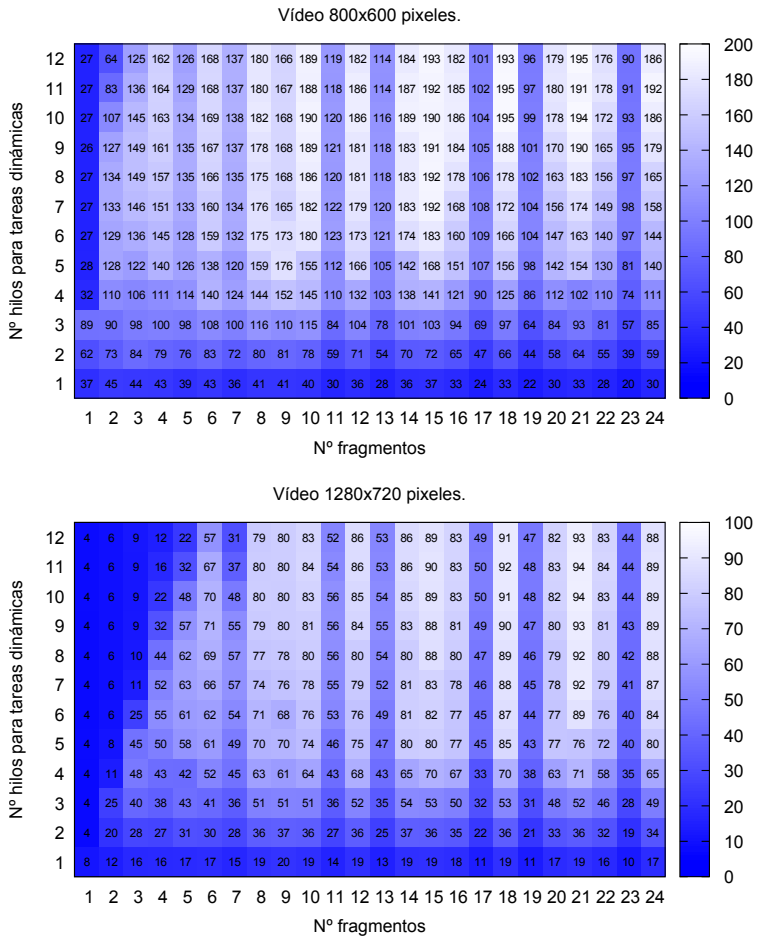

Figura 4. FPS para vídeos de 800x600 y 1280x720 píxeles

La memoria caché se comparte entre los cuadros que están siendo procesados en paralelo. Cuanto más cuadros se procesen en paralelo, mayor será la competencia por la memoria caché. La cantidad de cuadros que se procesan en paralelo está en relación con el número de fragmentos y la cantidad de hilos de ejecución para tareas dinámicas. Considerando que la cantidad de hilos de ejecución es mayor que la cantidad de fragmentos, si se disminuye la cantidad de fragmentos, manteniendo la cantidad de hilos de ejecución para tareas dinámicas, se incrementa la cantidad de cuadros a procesar en paralelo. Por lo tanto, en este caso, aumenta el número de fallos de caché.

Para comprobar que la hipótesis es correcta se diseñó el siguiente experimento. Se ejecutó el programa procesando cuadros de un vídeo de 1280x720 píxeles de resolución, y se midieron los fallos de caché en configuraciones de 1, 2, 6 y 12

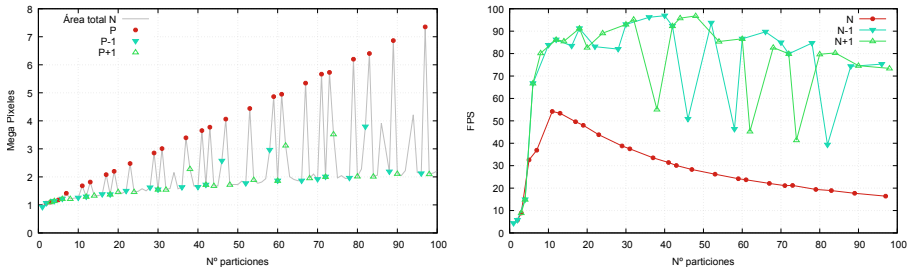

Figura 5. Comportamiento con números de fragmentos primos y sus inmediatos. A la izquierda el área total de la imagen, y la derecha los FPS.

fragmentos y cantidad de hilos de ejecución de tareas dinámicas de 1 a 11. Para obtener los fallos de caché de nivel 3 (es decir, cuando es necesario acceder a datos en la memoria RAM) se utilizó la herramienta *perf*. Ésta es una aplicación que permite acceder a los contadores de hardware del procesador que proveen datos muy precisos con un overhead despreciable.

En la figura 6 se muestran los fallos de caché por cuadro obtenidos en el experimento. Como se puede apreciar, a menor cantidad de fragmentos, mayor es la cantidad de fallos de caché ocurridos por cuadro.

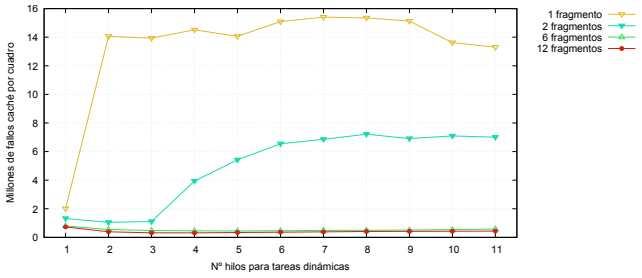

Figura 6. Fallos de caché por cuadro para distintas cantidades de fragmentos

Finalmente, en la figura 7 se muestran los tiempos máximos de retardo de los cuadros, para vídeos de 800x600 y 1280x720 píxeles de resolución (considerando que los FPS son limitados al máximo encontrado en la primera etapa de los experimentos). La figura 8 sintetiza la relación entre los FPS y el retardo de cuadros, mostrando el mínimo retardo de cuadros para los diferentes FPS que puede alcanzar el sistema en cada vídeo.

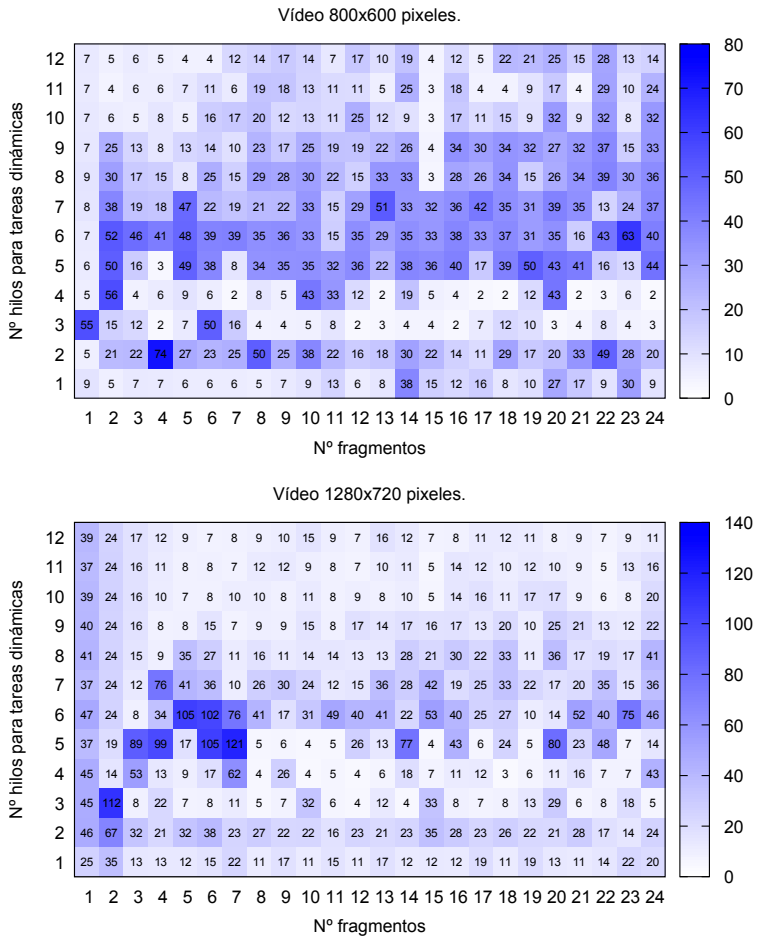

Figura 7. Retardo de cuadros para vídeos de 800x600 y 1280x720 píxeles

4. Conclusiones y trabajos futuros

En este trabajo se ha presentado un nuevo sistema de visión global por computadora para el fútbol de robots de la SSL, que puede ser utilizado como herramienta educativa en una asignatura de visión por Computadora y permite explorar distintas estrategias de paralelización. El sistema procesa cuadros de vídeo y reporta la posición y orientación de los robots y la posición de la pelota en cada uno de ellos. Se describieron dos estrategias de paralelización que son aplicadas en conjunto. Una de ellas explota el paralelismo dentro de cada cuadro, dividiendo los cuadros en fragmentos que son procesados de forma independiente. La otra estrategia se basa en el procesamiento simultáneo de diferentes cuadros del vídeo (independientes unos de otros). Se realizó una implementa-

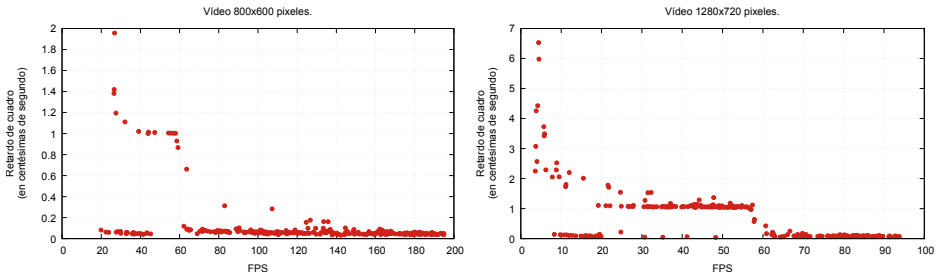

Figura 8. Retardo mínimo alcanzado para los FPS soportados por el sistema

ción en OpenMP para C++, basada en plugins para facilitar su modificación en un ambiente educativo. Se detallaron las tareas que ejecuta el sistema para implementar las estrategias de paralelización. Se estudió el comportamiento del sistema en términos de FPS y de retardo en el procesamiento de los cuadros para diferentes configuraciones del sistema en distintas resoluciones de vídeo. En un servidor con un procesador Intel Xeon E5-2630 (6 cores y multithreading simultáneo), la solución paralela respecto a la solución que utiliza un único núcleo (de los 6 disponibles) logró mejoras máximas de 5,27x en los FPS procesados mientras el retardo por cuadro se redujo a la mitad.

Como trabajo futuro se planea hacer una interfaz de usuario gráfica para configurar el sistema. Además, se cree conveniente implementar soluciones paralelas optimizadas de los plugins para utilizar el sistema en competencias de fútbol de robots.

Referencias

1. Small Size League Technical Committee. Laws of the RoboCup small size league 2015. `http://robocupssl.cpe.ku.ac.th/_media/rules:ssl-rules-2015.pdf`, March 2015. [Online 2016-03-10].
2. RoboCup. A brief history of robocup. `https://www.robocup.org/about-robocup/a-brief-history-of-robocup/`, 2016. [Online 2016-06-29].
3. Guillermo Torres, Javier Balladini, and Rodolfo del Castillo. Un sistema de visión global para fútbol de robots. 2014. Trabajo de tesis para optar al grado académico de Licenciado en Ciencias de la computación.
4. Stefan Zickler, Tim Laue, Oliver Birbach, Mahisorn Wongphati, and Manuela Veloso. Ssl-vision: The shared vision system for the robocup small size league. In *RoboCup 2009: Robot Soccer World Cup XIII*, pages 425–436. Springer, 2010.