
Segmentación y detección de objetos en FPGA

IPD432 - Diseño avanzado de sistemas digitales

Alexis Diomedi

22 de Marzo de 2017

1. Introducción

El procesamiento de imágenes y video en tiempo real siempre ha representado un desafío en términos computacionales. Dada las grandes cantidades de información que se han de manejar en períodos de tiempo restringidos, es importante la optimización de los algoritmos utilizados, y la utilización de hardware dedicado puede ayudar considerablemente a la ejecución de los mismos.

Particularmente, el problema de segmentación, siendo una de las primeras etapas para la extracción de información a partir de datos visuales, es también uno de los más importantes. Para este proyecto se planea abordar el problema del procesamiento de video para la segmentación de movimiento, lo cual es parte esencial de un sistema de seguimiento de objetos, y tiene aplicaciones áreas tales como sistemas de navegación autónomos, automatización de procesos de manufactura y sistemas de televigilancia.



Figura 1: Segmentación de peatones en una imagen. Cada *blob*, encerrado en un cuadro, representa una persona.

2. Descripción del sistema

Para el desarrollo de este proyecto se utilizó una placa de desarrollo *Atlys* de Digilent, la cual posee un chip Spartan 6 LX45. El diseño implementado en este proyecto se divide en 2 bloques principales. El

primero es el Substractor de fondo, el cual captura el flujo de video de entrada y compara pixel a pixel cuánta diferencia hay con el modelo de *background* almacenado en la memoria, y en función de eso se obtiene una máscara binaria de *Foreground/Background* para cada frame. El segundo bloque consiste en el módulo Analizador de *blobs*, el cual toma los datos de salida del bloque anterior y reconoce grupos de pixeles detectados como *foreground* para asignarlos a un *blob*, con lo cual procede a obtener el cuadro que encierra cada *blob*. Para propósitos de depuración y demostración, se agregaron una serie de multiplexores conectados a interruptores externos de la *Atlys* que permiten seleccionar la etapa del sistema que se quiere ver a la salida. Para mantener limitado el uso de recursos lógicos, se implementó un sistema que maneja flujos de video de 640 x 480 pixeles a 60 fps.

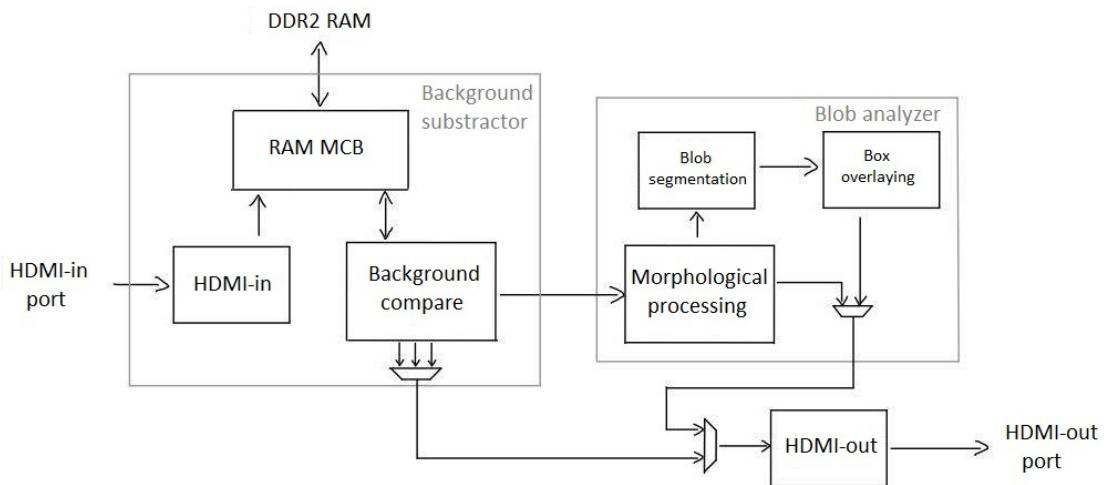


Figura 2: Diagrama de bloques del sistema

2.1. Background subtractor

El módulo subtractor de fondo puede dividirse en 3 submódulos: la interfaz de entrada HDMI, el MCB (Memory Controller Block) de la RAM externa, y el módulo de comparación y actualización del fondo.

La interfaz de entrada HDMI maneja toda la lógica de conexión y recepción de datos de video desde dispositivos externos, los cuales guardan en un buffer en la RAM externa a través del MCB.

El controlador de memoria (MCB) es un IP core generado con las herramientas de Xilinx ISE con parámetros ajustados para poder funcionar con el chip de memoria RAM DDR2 de 128 MB presente en la *Atlys*. Este consta de 4 puertos bidireccionales con un largo de palabra de 32 bits, los cuales, sin embargo, están configurados en este proyecto para funcionar en una sola dirección, 2 puertos de escritura que editan los buffer del frame de entrada y del *background*, y los otros 2 que leen de ambos buffer.

El último módulo es el que lee tanto los buffer de entrada como de *background* de la RAM externa y los compara línea a línea, obteniendo la distancia de Manhattan entre pixeles correspondientes para discriminar entre fondo y frente. Adicionalmente, esta incorporada en el bloque la lógica de actualización del fondo, que cada cierta cantidad de frames (controlable por un switch externo) reescribe el buffer de *background* según un promedio ponderado.

2.2. Blob analyzer

Al igual que el módulo anterior, este bloque también se divide en 3 secciones de lógica, consistentes en la fase de procesamiento morfológico, la segmentation de *blobs*, y la lógica de dibujo para los cuadros que encierran a cada blob.

La fase de procesamiento morfológico es básicamente la aplicación de un operador de apertura, consistente en una erosión seguida de una dilatación, a la máscara de *foreground* que entra. El propósito de esta parte es eliminar pequeñas agrupaciones de pixeles de la máscara de entrada, haciendo el sistema más robusto al ruido y eliminando la detección de falsos positivos.

La lógica de segmentación procesa los datos de la máscara pre-procesada para poder agrupar los grupos de pixeles de *foreground* en distintos *blobs*, según el algoritmo descrito en [1]. Esto permite obtener la *bounding box*, el cuadro que encierra a cada objeto presente en la pantalla. Por simplicidad se implementó un sistema capaz de reconocer hasta 15 *blobs* distintos.

Finalmente, al terminar cada frame, los datos de los bordes correspondientes a cada *blob* encontrado son guardados en registros, los cuales son leídos posteriormente para determinar las posiciones donde los pixeles corresponden a bordes de los cuadros y poder dibujarlos oportunamente.

3. Resultados

El sistema implementado es capaz de detectar objetos en tiempo real, los cuales señaliza encerrándolos dentro de un recuadro rojo.

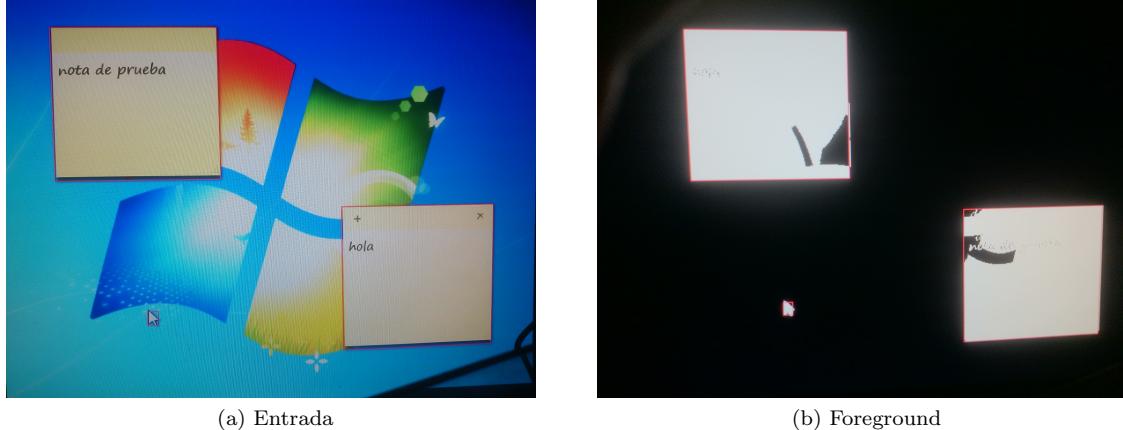


Figura 3: Imagen de entrada junto con su máscara de *foreground*. Notar que el sistema encierra en recuadros los objetos detectados

El switch 0 permite alternar la salida entre entrada y *foreground*, mientras que el switch 1 muestra el buffer de *background* guardado en la RAM. Note que las imágenes de fondo guardan un pequeño remanente de fondos anteriores, esto es debido al error de aproximación que se da al dividir para calcular el promedio ponderado que actualiza el *background*. El switch 2 permite acelerar el tiempo de actualización del fondo, y los switches 3 y 4 controlan la visualización de las salidas del *blob analyzer*.

Dado que tiene un número limitado de *blobs* hay ocasiones en que se observan objetos que no son detectados, esto es porque el sistema está configurado para ignorar los objetos adicionales que detecte una



Figura 4: *Background* contra el cual se compara la entrada de la figura 3

vez que alcanzó su límite.

La fase de apertura de analizador de *blobs* funciona bien al momento de filtrar puntos aislados, como se puede apreciar en la figura 5. Esto es particularmente útil en imágenes provenientes de cámaras con ruido, y ayuda a evitar una sobrecarga de *blobs* en la etapa posterior.

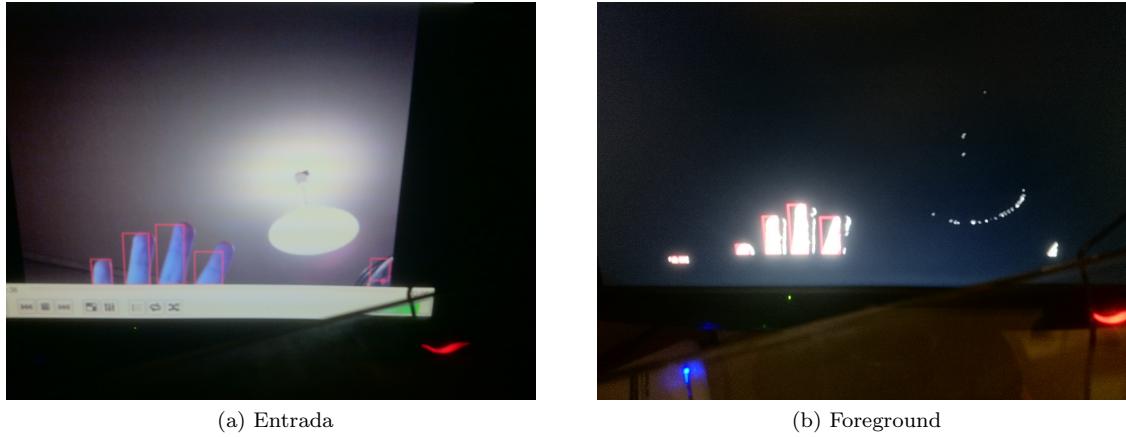


Figura 5: Detección de objetos en una imagen proveniente de una cámara

Cabe destacar que todas las pruebas hechas al módulo son hechas al mismo framerate (60 fps) donde, a diferencia del programa que correría en un computador, no se puede observar ninguna clase de enlentecimiento, y todo esto es logrado usando frecuencias de reloj consideradas extremadamente bajas para los computadores convencionales, siendo el reloj más rápido de 250 MHz y con gran parte del procesamiento hecho con un reloj de 25 MHz. Si bien por cuestiones de tiempo, relativas a la síntesis y mapeo del diseño, no se implementan módulos extra para mejorar el funcionamiento, tales como un pre-filtro o un operador de cierre, el código está lo suficientemente modularizado como para aplicar esos cambios fácilmente.

4. Posibles mejoras

Son varias las mejoras que se pueden implementar en el sistema descrito anteriormente. Para comenzar, la plataforma de hardware utilizada, a la fecha de escritura de este documento, ya se encuentra algo desactualizado, por lo que se podría considerar la implementación en una plataforma más moderna, para poder facilitar la expansión del diseño actualmente implementado. Otra de las mejoras más obvias es modificar el diseño para que sea capaz de soportar mayores resoluciones, o frame rates más altos. Cabe destacar también que existen partes del sistema que no necesariamente fueron implementados del modo más óptimo, y siempre hay lugar para mejorar.

Existen algunas ideas que surgieron en el desarrollo del proyecto pero no fueron implementadas. Una de ellas fue la aplicación de un filtro gaussiano para suavizar la imagen antes de hacer la substracción de fondo; esto facilitaría la detección de objetos pequeños con colores muy distintos al del fondo. Otra idea fue la implementación de un módulo que permita, vía puerto serial o usando los botones de la placa, modificar parámetros internos del sistema tales como el umbral para la detección de *foreground* o los elementos estructurantes de las etapas de erosión y dilatación. El modelo de *background* puede complejizarse aún más, implementando sistemas como los descritos en [3] y [4].

Es de particular interés la utilización de un sistema similar al presentado para acelerar tareas de procesamiento de imágenes en tiempo real, ejecutadas comúnmente por computadores y otras arquitecturas más clásicas; la FPGA podría entregar a un PC un flujo de video pre-procesado junto con información extraída que sirva para ahorrar tiempo de cómputo. La utilización de un sistema que combine ambas plataformas podría explotar el paralelismo y la baja latencia de una FPGA junto con la flexibilidad que entrega el software que corre en un computador.

Finalmente, dada la gran disponibilidad de pines y manejo de tan bajo nivel que disponen las FPGAs, una modificación que valdría la pena implementar sería reemplazar la entrada HDMI con una interfaz para comunicarse directamente con una cámara. Dado la flexibilidad que hay en cuanto a interfaces de hardware, la cámara a usar puede ser desde una simple webcam hasta una cámara multiespectral de alta complejidad.

Referencias

- [1] J. Trein, A. Schwarzbacher, B. Hoppe. *FPGA Implementation of a Single Pass Real-Time Blob Analysis Using Run Length Encoding*. IEEE Signal and Systems Conference, 2008.
- [2] J. Hiraiwa, E. Vargas, S. Toral. *An FPGA based Embedded Vision System for Real-Time Motion Segmentation*. 17th International Conference on Systems, Signals and Image Processing, 2010.
- [3] M. Genovese, E. Napoli. *ASIC and FPGA Implementation of the Gaussian Mixture Model Algorithm for Real-Time Segmentation of High Definition video*. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 22, No. 3, Marzo 2014.
- [4] Jia Wei Tang *et al.* *FPGA-Based Real-Time Moving Target Detection System for Unmanned Aerial Vehicle Application*. International Journal of Reconfigurable Computing, 2016.