Reconocimiento de objetos con GPU en Boya inteligente

Hector Rojas Stoll[1], Juan Tocino, Ezequiel Laurenti, Damian Vinci

[1]Universidad Nacional de La Matanza,
Departamento de Ingeniería e Investigaciones Tecnológicas,
Florencio Varela 1903 - San Justo, Argentina
Hector.rojass18@gmail.com, tocino.juan@gmail.conm, ezee.laurenti@gmail.com,
damianvinci@gmail.com

Resumen. Este trabajo de investigación intenta comprobar si es posible utilizar reconocimiento de objetos en nuestro proyecto de Boya inteligente a través de imágenes captadas por una camara integrada a la boya y de esta manera poder dar avisos en su correspondiente aplicación de android "BoyApp" sobre la presencia de objetos en una pileta, utilizando procesamiento en paralelo con GPU.

Palabras claves: Boya inteligente, BoyApp, HPC, GPU, OpenCV, Object Recognition, Reconocimiento de objetos.

1 Introducción

Esta investigación se basa en añadir a nuestro proyecto Boya inteligente, que consiste en realizar distintas mediciones tales como temperatura, opacidad o movimiento en una pileta que nos den una idea sobre el estado en tiempo real de la misma para poder tomar una decisión y realizar una acción de forma remota desde una aplicación movil o que la haga el embebido de forma automática y poder mantener siempre la pileta en optimas condiciones, una nueva funcionalidad que haga uso de computación de altas prestaciones (HPC) y base su solución en alguno de los paradigmas vistos en clase (OpenCL, MPI o GPU).

En nuestro caso, haremos uso del módulo GPU integrado en nuestro dispositivo Android, que a través del procesamiento de imágenes e información de forma paralela nos permitirá saber en tiempo real, si existen objetos en el fondo de la pileta brindandonos además información de los mismos, y a su vez, le da al usuario la posibilidad de seguir utilizando las funcionalidades convencionales de la app y la boya sin afectar la performance o tiempos de respuesta.

La motivación del desarrollo de esta nueva funcionalidad es por un lado saber sobre la acumulación de agentes externos que colaboran con la descomposición del agua tales como algunos insectos, moluscos o incluso hojas, sobre todo en epoca otoñal, y por otro lado la detección de objetos personales que a las personas se les suelen caer y perder

dentro del agua como por ejemplo joyas las cuales son importantes ubicar y recuperar rápido.

El tema del análisis de imágenes para la detección y obtención de informacion de objetos es un tema que se viene desarrollando desde hace tiempo y ya existen aplicaciones que cumplen estan funciones, por ejemplo una de las formas mas básicas de reconocimiento de imágenes es el codigo de barras que se comenzó a utilizar para el reconocimiento de objetos en almacenes e industrias y mas tarde se haría popular su uso en supermercados. Luego surgió el codigo bidimensional (QR) que permite almacenar mucha mas información que un codigo de barra, hasta llegar a hoy en ía en donde una de las aplicaciones más populares de reconocimiento de imágenes en dispositivos móviles es la aplicación de Google: Google Lens [1], la cual es capaz de, al hacer una fotografía, reconocer ciertas imágenes que aparecen en ella, pero está a diferencia de nuestro proyecto, sube la imagen a servidores en la nube donde se la procesa, se extrae información de ella y se hace una consulta en el motor de búsqueda al respecto pudiendo reconocer además de objetos, texto, códigos de barra y QR, etc.

2 Desarrollo

Para aplicar la funcionalidad propuesta en esta investigación, como primer medida, necesitaremos integrar a la boya una cámara como medio de obtencion de información, para que, de esta manera, paralelamente se pueda procesar la misma y llevar a cabo el reconocimiento del objeto.

Las imágenes captadas por la cámara son enviadas a la placa Arduino a la cual estará conectada, y luego serán enviadas al dispositivo móvil donde las imágenes seran procesadas y analizadas mediante el modulo GPU con el fin de detectar la presencia de algún objeto en el fondo de la pileta, este módulo además mediante algoritmos de reconocimiento de patrones y puntos característicos comparará esta información con otra almacenada en una base de datos que implementaremos y donde se encontrarán imágenes de posibles obejtos que se puedan encontrar en una pileta generalmente, con su respectiva información.

Finalmente, una vez reconocido el objeto, en el caso de tratarse de hojas o bichos solo se enviará una notificacion al dispositivo móvil avisando de la presencia del mismo, y en caso de tratarse de objetos personales o no identificados se procederá además a enviar la imagen del mismo acompañado de su información en caso de tenerla, para que el usuario sepa de antemano si solo necesita limpiar o el fondo o se trata de la detección de ese objeto perdido tiempo antes por él o alguien mas y realizar la acción correspondiente.

El diseño implementado permitirá que toda esta dinámica se haga en el dispositivo, sin tener que acceder a Internet para ello, y devolviendo resultados en un tiempo aceptable para el usuario, para lo cual utilizaremos algoritmos basados en el uso de técnicas para el reconocimiento de imágenes tales como puntos característicos o keypoints y descriptores.

Un punto característico es un lugar concreto de una imagen que contiene una propiedad que la define. Esto puede ser, por ejemplo, esquinas, bordes, formas u otros elementos que representen una particularidad de la imagen. Existen multitud de tipos de puntos característicos, entre ellos SIFT y SURF, que son los más extendidos, y otros como los BRIEF o KAZE que por sus características, son menos eficientes en su funcionamiento, aunque usados en ocasiones concretas [2]. Una vez extraídos los puntos característicos, se genera un descriptor para cada uno de ellos. Los descriptores son estructuras que contienen varios datos sobre el punto y su entorno, para poder definir un contexto para el mismo. Idealmente, estos descriptores deben ser invariantes a la escala, traslación o rotación de la imagen, así como cambios de iluminación o ruido. Otro de los tipos de punto y descriptor más conocidos son los tipo ORB (Oriented FAST and Rotated BRIEF) [3], la combinación de FAST y BRIEF, el cual podría plantarle cara a los puntos SIFT y SURF en cuestión de eficiencia y precisión.

En nuestro proyecto se utilizaran los puntos y descriptores ORB, ya que ofrecen ciertas ventajas:

1. Son más rápidos en su procesamiento, ya que sus descriptores son binarios.
2. En cuestión de precisión, son equivalentes a SIFT y SURF, que son los más extendidos.
3. ORB es de uso libre, mientras que SIFT y SURF operan bajo derechos de autor.

3 Explicación del algoritmo.

En primera instancia se realiza el emparejamiento de puntos, proceso mediante el cual los descriptores de dos imágenes son comparados y unidos en parejas. Existen varios métodos para este proceso, el más sencillo es el del vecino más próximo, dónde se calcula la distancia entre los descriptores y se emparejan con el más cercano.

Una vez se tienen los puntos emparejados, se busca una homografía que transforme los puntos de la primera imagen en la segunda. Una homografía es una transformación proyectiva entre dos figuras geométricas planas. En general se consideran que dos imágenes son equivalentes si existe dicha transformación. Para determinar los parámetros de la transformación, normalmente se utilizan sólo unas pocas parejas de puntos. La transformación resultante se aplica al resto de puntos, comparando el punto transformado con el punto emparejado. La homografía debe ser independiente del tamaño, localización y rotación del objeto de la escena. Este procedimiento es usado para comprobar qué cantidad de puntos de la imagen de entrada pertenecen a la imagen de la base de datos y si estos coinciden. Si se dispone de las suficientes parejas buenas, la imagen ha sido reconocida.

Para poder hacer programas utilizando los procedimientos descritos en este apartado, se utilizará la interface de Android de OpenCV [4], diseñado para la eficiencia computacional con fuerte enfoque en aplicaciones de tiempo real, del cual utilizaremos concretamente sus funciones relativas al tratamiento de imágenes (como pasar de una foto

en color a otra en escala de grises) y funciones para la extracción de puntos característicos, así como sus descriptores, emparejamientos, homografía y vecinos más próximos [5].

Además debido a que necesitamos paralelizar el trabajo para agilizar los tiempos de procesamiento, será necesario aplicar la arquitectura CUDA (Arquitectura Unificada de Dispositivos de Cómputo) [6], que hace referencia a una plataforma de computación en paralelo incluyendo un compilador y un conjunto de herramientas de desarrollo creadas por nVidia que permiten a los programadores codificar algoritmos en GPU [7].

El siguiente código ejemplifica el uso de CUDA con el algoritmo ORB.

```
//Se pasa la imagen a escala de grises y se sube a la GPU
cv::Mat img = cv::imread("image.jpg",
CV_LOAD_IMAGE_GRAYSCALE);
cv::cuda::GpuMat gpu_img;
gpu_img.upload(img);

//Se reduce el tamaño de la imagen leída tanto al eje x como y
para reducir el tiempo de procesado
cuda::resize(input, output, Size(), .25, 0.25, CV_INTER_AREA);

//Se define el descriptor ORB y el vector para los puntos
característicos

cv::Ptr<cv::cuda::ORB> orb = cv::cuda::ORB::create();
std::vector<cv::KeyPoint> keypoints;

//Se ejecuta el método de detección de puntos característicos
orb->detect(gpu_img, keypoints);
```

En este ejemplo se aplica paralelismo al trabajar con la imagen, lo cual será necesario ya que al ser imágenes tomadas bajo el agua se necesitará de mucha resolución de la cámara, lo que se traduce en más pixeles a analizar y por lo tanto más carga computacional, y luego se utiliza el algoritmo ORB para detectar los keypoints necesarios para realizar la comparación con otra imagen.

En cuanto a la GPU utilizaremos una configuración de bloques-hilos de (32x32) y una configuración de grilla-bloques de (69x69) por lo cual tendremos 1024 hilos por bloque y 4761 bloques por grilla, y la cámara que necesitaremos tendrá una resolución 4K (3.840x2.160 pixeles), es decir un total de 8.294.400 pixeles.

Ya que las imágenes serán procesadas por el GPU pixel a pixel, se trabajara cada pixel en un hilo distinto, por lo cual en cada bloque de la grilla puedo procesar 1024 pixeles de mi imagen, lo que nos dara un total de 8100 bloques y 2 grillas para la totalidad de la imagen.

4 Pruebas que pueden realizarse

Las pruebas que podrían realizarse para verificar el correcto funcionamiento de esta función implican arrojar diferentes objetos al fondo de la pileta, ya sean las que están cargadas en nuestra BD como otros que no lo estén, a partir de esto la boya debería detectar la presencia de los mismos enviando la correspondiente notificación al dispositivo móvil con un aviso, una imagen del objeto con o sin información adicional según corresponda.

5 Conclusiones

A través de esta investigación intentamos demostrar que se puede implementar el reconocimiento de objetos mediante la integración de una cámara en nuestro proyecto Boya inteligente, la utilización de las funciones de la librería OpenCV para Android y el modulo GPU de nuestro dispositivo móvil.

A partir de esta experiencia pudimos aprender sobre los grandes beneficios que conlleva la utilización de HPC a la hora de desarrollar un software que sea paralelizable, con alta demanda de poder de cómputo, con algoritmos muy complejos o neceidad de mucha memoria o ancho de banda, concluyendo que es una solución viable en cuanto a lo tecnológico y económico para tomar en cuenta en futuros proyectos.

Una mejora a futuro podría ser optimizar la función de comparación de la imagen recibida con las imágenes de la BD, ya que mientras la cantidad de imágenes sea baja no habrá problema, pero si esta empieza a crecer puede que se tarde un tiempo en realizar el proceso, para ello en vez de llamar a la función de comparación por cada imagen, arrancaría un hilo, que sería el encargado de realizar la tarea. Por tanto, se lanzarían tantos hilos como imágenes contenga la base de datos.

6 Referencias

1. Google Lens, https://lens.google.com/ [1].
2. Ebrahim Karami, Siva Prasad, Mohamed Shehata: Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison. St Johns, Canada (2015) [2].
3. Ethan Rublee, Vincent Rabaud, Kurt Konolige, Gary Bradski.: ORB: an efficient alternative to SIFT or SURF (2015) [3].
4. OpenCV, https://opencv.org/android/ [4].
5. I. Culjak, D. Abram, T. Pribanic, H. Dzapo, M. Cifrek: A brief introduction to OpenCV. Publisher: IEEE (2015) [5].
6. CUDA, https://developer.nvidia.com/cuda-toolkit [6].

7. F. Aguilera, J. Aceves, S. Arguelles, E. Gomez, G. Martinez: Utilizacion de GPU-CUDA en el Procesamiento Digital de Imágenes (2015) [7].