Revista Digital del Departamento de Ingeniería e Investigaciones Tecnológicas de la Universidad Nacional de la Matanza
ISSN: 2525-1333
Vol.: 5 - Nro. 1 (AGOSTO-2020)

Artículo original

DESARROLLO DE UN SISTEMA DE ASISTENCIA VISUAL PARA UN MANIPULADOR AUTOMÁTICO PROGRAMABLE (ROBOT INDUSTRIAL)

VISUAL ASSISTANCE SYSTEM FOR A PROGRAMMABLE AUTOMATIC HANDLER (INDUSTRIAL ROBOT)

Ing. Martín Ferreyra Biron, Ing. Alberto Miguens, Ing. Fernando Szklanny
Departamento de Ingeniería e Investigaciones Tecnológicas, Universidad Nacional de La Matanza

ISSN: 2525-1333. Vol.: 5 - Nro. 1 (AGOSTO-2020)

Resumen:

En el presente trabajo se describe el desarrollo de un sistema de procesamiento digital de imágenes y la adaptación de un algoritmo detector de objetos, utilizando cámaras de alta velocidad con el objetivo final de incorporar dicho sistema a un autómata programable, apto para moverse con seis grados de libertad de forma articulada y con el fin de ser utilizado en aplicaciones industriales con ciertas exigencias de eficiencia. El autómata, que se encuentra funcionando casi en su totalidad, está conformado por tres elementos: un manipulador, un controlador y un conjunto de sensores.

El controlador es el encargado de procesar la información para mover el manipulador y las unidades de control de energía. Por otra parte, el brazo está comandado por un servomecanismo que permite mover al mismo articuladamente en seis ejes, y por último los sensores utilizados, con la finalidad de mejorar el control del autómata y la seguridad en el entorno de trabajo, son cámaras de alta velocidad y acelerómetros.

El objetivo principal del presente consiste en el desarrollo del sistema de visión que dará soporte al autómata descripto.

Abstract:

This document describes the development of a digital image processing system and an object detector adaptation using high-speed cameras, with the final goal of being included into a programmable automat, suitable to move in six degrees of freedom, and oriented to industrial applications with specific efficiency requirements. This robot, currently working almost entirely, includes three elements: a manipulating system (the arm), a controller and a quantity of sensors.

The controller is the device that process the movement information and the energy control units. The arm is moved by a servomechanism and is capable of moving in six axis and, finally, the sensors required, in order to get a better control of the robot and to ensure safety in the work environment, are high-speed cameras and accelerometers.

In other terms, the main objective of the current work consists in the development of a visual system that will support the abovementioned robot.

Palabras Clave: *Visión, robot industrial, velocidad, seguridad.*

Key Words: *Vision, Industrial robot, speed, safety.*

Revista Digital del Departamento de
Ingeniería e Investigaciones
Tecnológicas de la Universidad
Nacional de la Matanza
ISSN: 2525-1333. Vol.: 5 - Nro. 1 (AGOSTO-2020)

I. INTRODUCCIÓN

La aplicación de los sistemas digitales de procesamiento de señales y su aplicación al control de motores de inducción y autómatas programables tiene su origen en la década de 1970, con los primeros desarrollos de procesadores digitales de señales DSP por Intel, AMI y Bell Labs. Las dificultades en cuanto a la capacidad de procesamiento y velocidad hicieron que su aplicación se limite a controles de baja complejidad. Sin embargo, el avance tecnológico producido desde ese momento hasta hoy, hace factible aplicar estos sistemas digitales de procesamiento a controles con algoritmos de alta complejidad.

El autómata propuesto es un brazo mecánico capaz de moverse articuladamente en seis ejes, del tipo de "articulación coordinada", llamado así por la semejanza de sus movimientos con los del cuerpo humano. La programación de los movimientos del autómata se realiza desde una computadora personal mediante un software a desarrollar. Los autómatas programables se utilizan ampliamente en la industria automotriz, aunque existen numerosos sistemas de producción que no cuentan con autómatas que se adapten a sus necesidades, por lo que el trabajo que aquí se describe permite proponer soluciones innovadoras que cubran dichas necesidades. En particular se hace indispensable el desarrollo de algoritmos para el guiado de autómatas en base a la visión, lo que permitirá utilizarlos para detectar y manipular objetos de diversas índoles, mejorar la precisión al realizar una trayectoria, y evitar objetos no previstos en una tarea, haciendo al autómata adaptable a diferentes condiciones de trabajo.

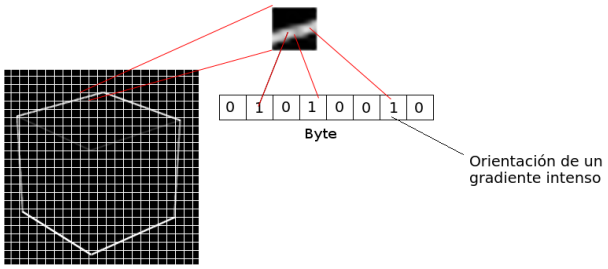

Fig. 1. Ejemplo de cómo se realiza un “template”

En conclusión, con el fin de mejorar la productividad de las industrias, la calidad y el costo de fabricación para ganar competitividad a nivel global, se hace interesante contar con autómatas programables en los procesos de fabricación con sistemas de visión.

II. MARCO TEÓRICO

El procesamiento digital de imágenes ha tomado un gran impulso en la última década y si bien en la actualidad existe una gran cantidad de bibliografía sobre estudios y algoritmos desarrollados para el control de motores de inducción, sistemas de control electromagnéticos, vehículos eléctricos, procesamiento digital de imágenes, entre otros, no abundan sistemas de producción que cuenten con autómatas que se adapten a sus necesidades respecto a procesos automatizados orientado a un aumento del nivel de producción y calidad y menos aún que utilicen el procesamiento digital de imágenes. A partir de esta situación se decidió desarrollar un brazo robot que pueda ser automatizado y tomar decisiones en sus movimientos, valiéndose de la entrada que proporcionan cámaras de video. Para que el brazo pueda tomar decisiones necesitará reconocer objetos y esto implica la adecuada elección de un

ISSN: 2525-1333. Vol.: 5 - Nro. 1 (AGOSTO-2020)

algoritmo que cumpla con tal fin. Al inicio de la investigación sobre estos algoritmos se hizo un estudio empírico sobre SURF[1] y ORB[2] los cuales forman parte de conocidos detectores aunque no se lograba una correcta detección en objetos que no tuvieran textura. Tampoco se deseaba utilizar redes neuronales convolucionales [3] debido al costo de procesamiento y de entrenamiento más allá de su buen desempeño en reconocer patrones y la atención [4], junto con las redes neuronales recurrentes no se encontraban en ese momento tan difundidas en el campo del procesamiento digital de imágenes, por lo tanto de todas las opciones estudiadas la que más se acercaba a nuestros requerimientos fue el algoritmo DOT [5] que fue el implementado y el cual se detalla en los siguientes párrafos.

III. IMPLEMENTACIÓN DE DOT

Como se comentó en párrafos anteriores, parte del proyecto implica el desarrollo e implementación de DOT, algoritmo del cual daremos una breve reseña.

A. DOT

DOT es un algoritmo de detección de objetos presentado en la CVPR 2010 el cual tiene la capacidad de poder detectar objetos con o sin textura de una manera rápida y repetitiva valiéndose de distintas imágenes procesadas del objeto a buscar, a las que se denomina patrones (a partir de ahora se utilizará la terminología original "*templates*"). DOT se puede resumir en tres grandes pasos:

- Creación de los "templates".
- Preparación de la escena.
- Búsqueda del objeto.

B. CREACIÓN DE TEMPLATES

La primera etapa es la creación de los "templates" a partir de los objetos a buscar, siendo la metodología propuesta por DOT simple: obtener imágenes del objeto desde distintas perspectivas para luego procesarlas y obtener finalmente los "templates". Mientras más "templates" distintos se obtengan, más eficiente será la búsqueda, pero también más lenta. Estas imágenes serán la materia prima de los "templates" y se buscarán posteriormente en la escena. Luego, a cada imagen del objeto obtenida se le aplica el filtro Sobel [6] para así poder hallar los gradientes de la imagen. La imagen resultante se divide en "pequeños" cuadrados de N x N pixel y se recogerá la orientación de los primeros 7 gradientes más intensos en cada cuadrícula. Estos datos se guardarán en un byte siendo el MSB el bit que indique si en esa celda de la cuadrícula hubo o no gradientes.

C. PREPARACIÓN DE ESCENA

Se denomina "escena" a la imagen en la que se buscará el objeto a detectar. El proceso es similar a como se prepara un "template": los bordes de la escena son detectados con el filtro Sobel para así obtener los gradientes, luego se divide la imagen en cuadrados de la misma medida utilizada en los "templates" con la diferencia de que no se buscarán los 7 gradientes más intensos por cada cuadricula, sino que se reservará únicamente la orientación del gradiente más intenso.

D. BÚSQUEDA

Finalmente, la etapa de búsqueda, conceptualmente sencilla, consta en "deslizar" sobre la escena cada "template" calculando cuántas orientaciones

ISSN: 2525-1333. Vol.: 5 - Nro. 1 (AGOSTO-2020)

de los gradientes más intensos de la escena coinciden con las orientaciones de los "templates". En el lugar donde haya mayores coincidencias es donde se supondría que se encuentra el objeto a detectar.

IV. IMPLEMENTACIÓN DE DOT EN GPU

Si bien la implementación propuesta en el trabajo de Hinterstoisser fue realizada en una computadora con arquitectura x86 sin GPU, la performance obtenida no fue la esperada para poder aplicar esta técnica a un sistema de visión de un brazo robot, por lo tanto, con el objetivo de avanzar en el desarrollo del algoritmo, se procedió a la incorporación al sistema de una placa de desarrollo Nvidia Jetson TK1 la cual posee una GPU.

En un primer intento se compiló el código realizado anteriormente con mínimas modificaciones para adaptarlo a la placa mencionada. Las consideraciones iniciales, sumado a que gran parte del código hace uso de la biblioteca OpenCV, [8] y que varios métodos utilizados están optimizados para la ejecución en una GPU, sugerían un buen rendimiento, pero esto no fue así. La detección en un video de prueba superaba por mucho el tiempo de procesamiento al de su ejecución mediante el análisis en una CPU. Observando este inconveniente se comenzó a analizar el código con detenimiento. Se observó que la manera en la que se estaba intentando utilizar la GPU no era la adecuada, por lo tanto, se examinó cada uno de los métodos utilizados para identificar aquellos que producían la mayor demora y optimizarlos. A partir de este análisis se llegó a la conclusión de que el cálculo de gradientes en la escena era el primero que más demoraba en ejecutarse. Recordemos que el método se debe ejecutar fotograma a fotograma ya que se trata de un video. En el cálculo de gradientes las tareas a realizar son las siguientes y cada una de ellas depende de la anterior:

1. Se convierte a escala de grises la imagen del fotograma a analizar.
2. Se calcula el filtro Sobel por X e Y.
3. Se calcula el gradiente a partir de los filtros anteriores.
4. Se halla el máximo gradiente en cuadrados de NxN pixeles en toda la imagen.
5. Se calcula el ángulo de dicho gradiente guardando su orientación en un byte.
6. Se almacena el resultado.

De toda la lista anteriormente mencionada, el primer lugar en donde se atacó la demora fue en el cálculo del filtro Sobel. Se intentó utilizar, nuevamente, pero de otra forma, el filtro Sobel optimizado para CUDA ofrecido en OpenCV y ejecutado en paralelo (para obtener los componentes X e Y) pero los tiempos globales de detección nos hizo pensar que la mejor forma de ganar rendimiento al implementar el algoritmo DOT no era utilizando los métodos optimizados ofrecidos por OpenCV, sino a través del desarrollo de un kernel. Esto fue producto de dos circunstancias, en un principio desconocidas, al momento de iniciar las pruebas:

- El tiempo de transmisión de datos desde la memoria principal a la memoria de la GPU (y viceversa) no es despreciable y de manera inherente genera una sobrecarga de tiempo en la ejecución del algoritmo. Estos tiempos deben ser minimizados.

ISSN: 2525-1333. Vol.: 5 - Nro. 1 (AGOSTO-2020)

- Se debe buscar en la imagen máximos locales en regiones de interés de NxN pixeles ya que OpenCV no ofrece una función optimizada que utilice CUDA para realizar esta tarea.

Por todo lo expuesto se procedió al diseño de un kernel que pudiera acelerar y hacer uso correcto de la GPU. y así hallar los gradientes eficientemente. El enfoque utilizado para resolver el problema fue dividir y solucionar el mismo en etapas parciales.

El primer paso fue el desarrollo de un kernel que permitiera aplicar el filtro Sobel a una imagen utilizando la GPU. Esto fue fundamental ya que en un kernel se solucionaban los puntos 2 y 3 de la lista de tareas mencionada. A continuación, se ensayó el comportamiento del kernel sobre un video y los resultados fueron alentadores: en un tiempo mucho menor de lo esperado, el filtro era aplicado. Por lo tanto, con estos cambios la lista de tareas quedaría de la siguiente forma:

1. Se convierte a escala de grises la imagen del fotograma a analizar.
2. Se calculan el filtro Sobel por X e Y y el gradiente (GPU).
3. Se halla el máximo gradiente en cuadrados de NxN pixeles en toda la imagen.
4. Se calcula el ángulo de dicho gradiente guardando su orientación en un byte
5. Se almacena el resultado.

El paso siguiente consistió en la búsqueda del máximo gradiente en los bloques de NxN pixeles en la GPU. La primera aproximación para solucionar este problema fue programar en un kernel distinto la búsqueda de máximos locales utilizando la GPU en bloques de NxN pixeles. De esta forma se trabajó la imagen resultante del filtro anterior en cuadrados NxN, pudiéndose hallar paralelamente los máximos de cada

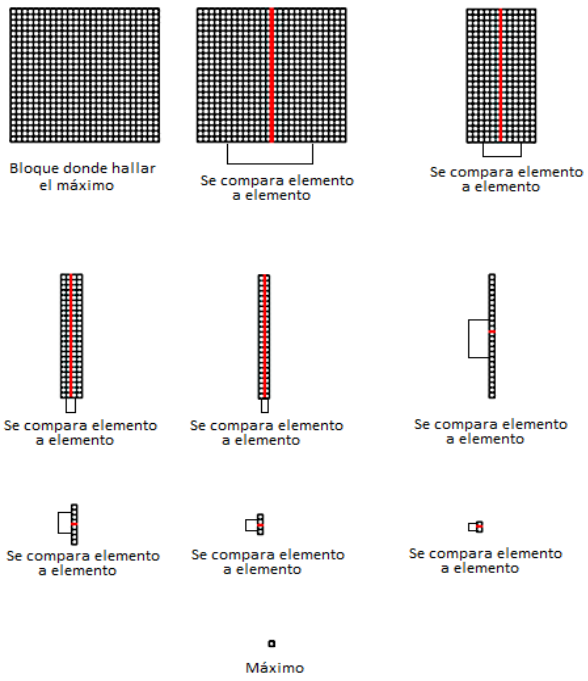

Fig. 2. Ejemplificación de la reducción de una matriz utilizando una GPU

bloque. Esto fue correcto, pero el hecho de utilizar un único hilo para buscar el máximo gradiente desperdiciaba el poder de procesamiento de la GPU. Es por esto por lo que después de analizar el problema se reprogramó la solución utilizando el concepto de reducciones paralelas, en este caso, en matrices.

La idea de las reducciones paralelas es sencilla (la figura 2 ejemplifica de una manera clara el concepto de reducción que se programó): para hallar un máximo se deben utilizar comparaciones. Si se utiliza un solo núcleo o hilo, ese único hilo o núcleo debe realizar todas las comparaciones. Pero al tener una GPU y poder decidir

ISSN: 2525-1333. Vol.: 5 - Nro. 1 (AGOSTO-2020)

cuantos hilos se ejecutan por cada bloque, esta situación cambia: como la operación de comparación es binaria, la máxima cantidad de hilos que pueden utilizarse para solucionar el problema de manera óptima es la mitad de la cantidad de elementos (si esta es par). Por ejemplo, si se trata de hallar el máximo en una matriz de 8x8 elementos existirán 64 elementos a comparar y si se utilizan 32 hilos se podrán comparar 32 pares de elementos de una sola vez, guardando el resultado en una de las mitades. Si esta operación se continúa realizando (con los elementos que sobran comparar) la eficiencia que se obtiene es realmente considerable.

A posteriori de haberse hallado el máximo, en el mismo kernel se programó el cálculo para transformar y expresar su orientación medida en un byte, tal como lo proponen los autores del algoritmo DOT. Por lo tanto, la lista de tareas con los cambios realizados se resume en:

1. Se convierte a escala de grises la imagen del fotograma a analizar.
2. Se calculan el filtro Sobel por X e Y, y el gradiente (GPU)
3. Se halla el máximo gradiente en cuadrados de NxN pixeles en toda la imagen, se calcula el ángulo de dicho gradiente guardando su orientación en un byte y se almacena el resultado (GPU)

Las optimizaciones descriptas producen un considerable aumento de velocidad respecto a las mismas tareas ejecutadas en una CPU o utilizando en parte la GPU, a través de OpenCV.

Ya solucionado en gran parte la demora producida por el cáculo de los gradientes, se continuó analizando el programa en búsqueda de otras demoras. Un punto clave que producía las mismas, fue la decodificación del método de compresión utilizado en los videos o en la misma cámara de la que se obtienen los fotogramas. Debido a que la única manera de aumentar la velocidad de esta decodificación es utilizar la GPU, y la transferencia entre la memoria principal y la de la GPU produce demoras, se decidió decodificar los fotogramas en un hilo común en la CPU.

Se debe tener en cuenta que la placa Nvidia Jetson TK1 posee un microcontrolador ARM de cuatro núcleos y utilizar uno en particular para decodificar los fotogramas no produce ningún tipo de problemas. Valiéndose de una arquitectura de productor consumidor, en el que en un hilo se decodifican los fotogramas (productor) y en otro hilo se ejecuta el kernel donde se buscan los gradientes y los máximos (consumidor), se aumentó aún más la eficiencia de ejecución del algoritmo DOT.

A. OPTIMIZACIÓN DE LA BÚSQUEDA

En este punto se entiende que la GPU ya no puede ser sobrecargada, por lo tanto, la búsqueda del objeto se debe realizar en la CPU. El algoritmo DOT propone que para encontrar un objeto es necesario buscar la coincidencia más probable entre la escena y los objetos que se aprendieron. Para lograr esto se necesita utilizar una ventana deslizante (que recorre toda la escena) y buscar coincidencias de ángulos de los objetos aprendidos. El “ganador” o la región candidata a ser el objeto en cuestión es aquella en la que existan más coincidencias de orientaciones entre la plantilla del objeto aprendido y la escena. En una versión simplificada del código se vería así:

Revista Digital del Departamento de
Ingeniería e Investigaciones
Tecnológicas de la Universidad
Nacional de la Matanza
ISSN: 2525-1333. Vol.: 5 - Nro. 1 (AGOSTO-2020)

```
Por cada plantilla de objeto aprendido
    Por cada pixel en Y de la escena
        Por cada pixel en X de la escena
            Por cada pixel en Y de la plantilla
                Por cada pixel en X de la plantilla
                    Match+=Escena(x,y)&plantilla(x,y)
```

La alta complejidad computacional salta a la vista y de alguna manera habría que minimizarla. Luego de analizar el algoritmo se llegó a las siguientes conclusiones: se podría aumentar la eficiencia si solamente se comprobara la coincidencia de ángulos en los lugares donde estén definidos en la plantilla, de esa manera se evitaría una iteración "*for*" y, por ende, varias operaciones en la búsqueda. Además, se notó, experimentalmente, que es mucho más rápido hacer la búsqueda cada dos pixeles en la escena que uno por uno (sacrificando resolución en la búsqueda por velocidad).

Para lograr esto se modificó el método que utiliza el programa de aprendizaje para analizar y guardar los objetos aprendidos, ahora en vez de guardar toda la plantilla se guarda solamente el valor del/los ángulos y la posición en la misma. En resumen, una versión simplificada del código se vería así

```
Por cada plantilla del objeto aprendido
    Por cada pixel par en Y de la escena
        Por cada pixel par en X de la escena
            Por cada ángulo válido en la plantilla
                If(Escena(x,y) es un ángulo definido)
                    Match+=Escena(x,y)&plantilla (x,y)
```

Si bien el cambio no es drástico, es evidente que existen mejoras.

B. CAMBIOS DRÁSTICOS EN EL OBJETO

El algoritmo DOT no es perfecto y en las pruebas se observó que quizás por falta de plantillas la detección no se realizaba en el lugar correcto. Para morigerar este comportamiento se intentó suavizar la detección brusca de objetos, teniendo en cuenta que estos no pueden aparecen en otro lugar en la escena de instantáneamente. Esta situación se trató de solucionar de la siguiente manera: si se detectó un objeto en una posición que supera un umbral de detección, la ventana de detección se trasladará de manera progresiva al lugar donde se supone que se encuentra el objeto.

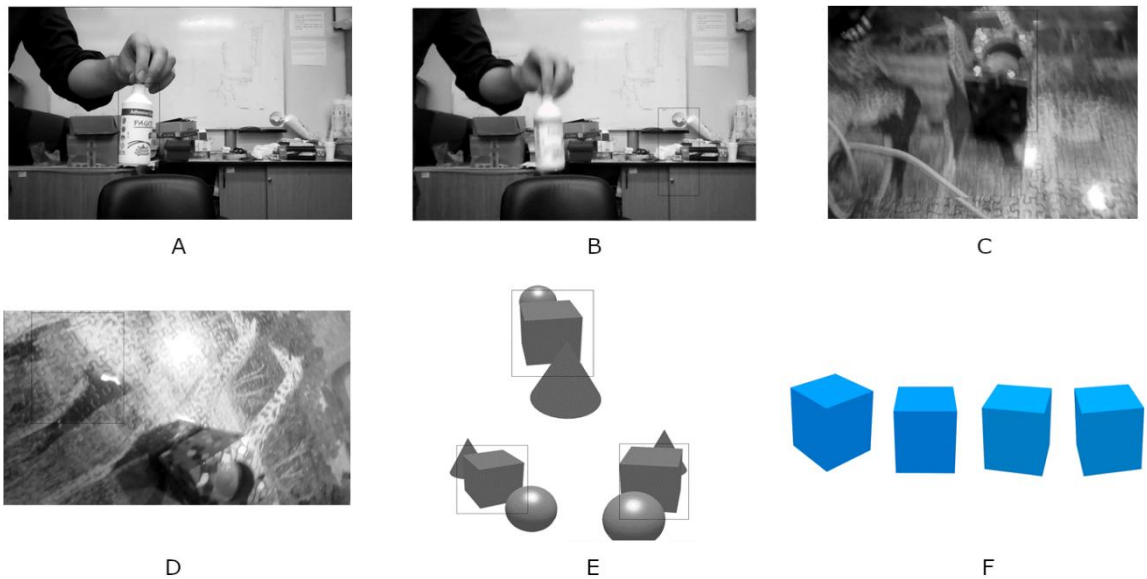

Fig. 3. Imágenes resultantes del detector. A) Algoritmo reconociendo el envase de pegamento. B) Algoritmo presentando una falla en la detección C) Algoritmo detectando un cubo correctamente D) Algoritmo fallando por falta de plantillas. E) Detecciones correctas en el video de prueba que se utilizó. Observar la detección con oclusiones. F) Plantillas que se utilizaron para la detección de la imagen E.

ISSN: 2525-1333. Vol.: 5 - Nro. 1 (AGOSTO-2020)

C. PRUEBAS

En la figura 3 se presentan algunas imágenes de momentos en los que el algoritmo produjo resultados apropiados. Es importante recalcar que fueron algunos momentos, debido a que no se utilizaron demasiadas plantillas para realizar la detección. Las imágenes son grises debido a que se consume demasiada memoria si se utilizan los tres canales de colores. El exceso de consumo de memoria genera, como consecuencia, que el sistema operativo termine de manera abrupta el proceso de detección.

V. DETECCIÓN DE PROFUNDIDAD

La detección de un objeto a partir de la implementación de DOT permite encontrar el mismo en el plano, pero no en el espacio, siendo parte fundamental la detección de esta dimensión debido a la naturaleza posicional del brazo robot. Es por esto por lo que se necesita realizar la medición de la profundidad por medio de un sistema estereoscópico. La posibilidad que se evaluó es la de ubicar dos cámaras en un mismo plano *M* (con coordenadas *x* e *y*) separadas por una distancia *b* en el eje *x*, y con sus ejes focales paralelos y perpendiculares al plano *M*. De esta forma se lograrían dos imágenes dispares a nivel horizontal, y el producto de esa disparidad se utilizaría para medir la profundidad.

A nivel matemático [9], este mecanismo puede ser resuelto a través del método de triangulación. En la Figura 4 se puede observar un esquema básico, en el cual se muestran dos sistemas de ejes cartesianos cada uno correspondiente a cada cámara del par estereoscópico.

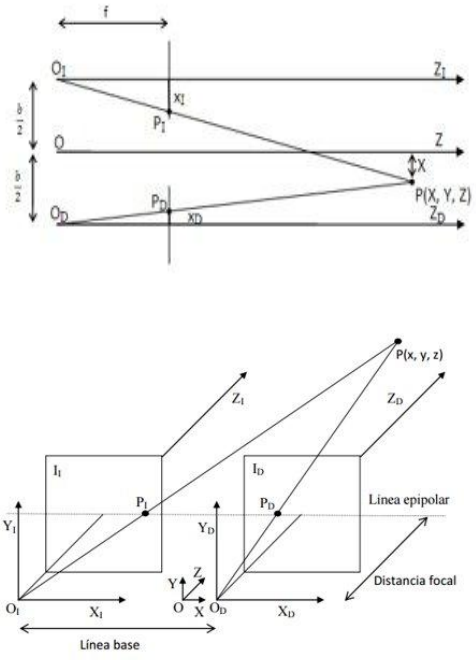

Fig. 4. Diagrama tridimensional para el cálculo de la distancia del punto P

A una profundidad dada por la distancia focal de las cámaras *f* (que debe ser la misma en ambas) se encuentra el plano denominado epipolar que contiene los planos I_I e I_D correspondientes a las imágenes tomadas por cada cámara. En este plano es donde se miden las disparidades entre dichas imágenes.

Al encontrarse las cámaras a la misma altura (eje y), las únicas diferencias se dan a nivel horizontal ya que, como se comentó, las cámaras se ubican a una distancia de separación *b* sobre el eje *x*.

En la Figura 5, se pueden observar los puntos P_I y P_D que son las proyecciones del punto P sobre la línea epipolar. Se puede intuir que cuanto más lejano se encuentre el punto P, los vectores de proyección tenderán a ser paralelos al eje Z lo que hará que P_I y P_D tiendan a ubicarse sobre el eje Z_I y Z_D. La disparidad del punto se

ISSN: 2525-1333. Vol.: 5 - Nro. 1 (AGOSTO-2020)

calcula como X_I - X_D = d y en el caso de que un objeto se encuentre ubicado en el infinito del eje Z hará que d valga 0. Así entonces el cálculo de la distancia del objeto sobre el eje Z será:

$$ImagenIzquierda: \frac{\frac{b}{2}+x}{z} = \frac{x_I}{f} \Rightarrow x_I = \frac{\left(x+\frac{b}{2}\right)f}{z}$$

$$ImagenDerecha: \frac{\frac{b}{2}-x}{z} = \frac{x_D}{f} \Rightarrow x_D = \frac{\left(x-\frac{b}{2}\right)f}{z}$$

Fig. 5. Diagrama bidimensional para el cálculo de la distancia del punto P

En la ecuación *Imagen Derecha* se asigna un signo negativo a X_D para compatibilizar los sistemas de coordenadas de los ejes O_I y O_D. Siendo la disparidad del punto P: x_I-x_D= d reemplazando y despejando se llega a la ecuación:

$$z = \frac{fb}{d}$$

con la cual se puede calcular la distancia z del punto P (objeto detectado).

Por otra parte, el desarrollo de los algoritmos de control del autómata se apoyó en las bibliotecas desarrolladas por Peter Corke, realizándose simulación mediante el software MATLAB. Queda pendiente la integración con las bibliotecas que se desarrollaron para la detección de objetos, debido a que se priorizó el desarrollo de las de procesamiento gráfico.

A. CALIBRACIÓN DE LAS CÁMARAS

Lo aconsejable fue, en un principio, que el ajuste se realizara en forma mecánica logrando la mejor alineación posible de las cámaras (ejes horizontal y vertical) para que luego el algoritmo de calibración tuviese mejores resultados. Una vez realizado este ajuste, se comenzó con el cálculo de las matrices de calibración, para lo que se utilizó la biblioteca de OpenCV que contiene clases para calibración y reconstrucción de imágenes 3D: "Camera calibration and 3D reconstruction". Para la calibración se usó la aplicación *calibrate_cameras*, que forma parte de la biblioteca StereoVision [10] (Bajo licencia pública GNU).

Esta aplicación toma como argumentos imágenes estéreo que contienen en su escena un patrón cuadriculado símil a un tablero de ajedrez, con capturas en distintas perspectivas de éste (Figura 6). El tablero se rota y, además, se lo traslada a lo largo de las múltiples

Fig. 6. Vértices detectados por el algoritmo de calibración

tomas de imágenes. Matemáticamente se puede demostrar que la cantidad mínima de tomas depende de la cantidad de esquinas que contiene el tablero patrón. El término "esquinas" hace referencia a la cantidad de vértices de cuadrados negros que hacen contacto entre sí (léase negros o blancos). La máxima cantidad utilizada es 50, bajo el criterio de que cuantas más muestras haya los cálculos de corrección de errores serán más precisos.

La aplicación *Calibrate_cameras* da como resultado una carpeta con un conjunto de matrices que contiene los parámetros intrínsecos (focos, centrados), extrínsecos (traslación y rotación) y las matrices de rectificación de distorsiones. Con estas matrices se realizan las correcciones sobre las imágenes tomadas en cada cámara. Una vez rectificadas las imágenes se puede realizar el cálculo del eje Z con la fórmula citada más

arriba. Para lograr este cálculo hay que ubicar en ambas imágenes del par estéreo el mismo objeto, para esto es necesario un algoritmo de búsqueda e identificación de objetos. Esto se puede resolver con el algoritmo DOT que fue detallado anteriormente.

B. *VERIFICACIÓN CON MAPA DE DISPARIDAD*

La verificación con mapa de disparidad se realizó utilizando las bibliotecas de StereoVision, una prueba empírica realizando un mapa de disparidad y una nube de puntos. La nube de puntos consiste en conseguir las coordenadas espaciales de cada pixel de la escena tomadas por las cámaras.

De esta forma se puede visualizar una escena desde diferentes perspectivas rotando los ejes de coordenadas. Esto tiene un límite y es que esos "puntos" de la imagen se obtienen desde una posición fija del espacio, si se deseara observar perspectivas laterales se necesitaría más información de la escena. Esto se puede salvar hasta cierto ángulo, replicando los pixeles de la frontera, de forma que cubran los huecos de información faltante.

Para lograr un buen mapa de disparidad se necesita que los objetos de la escena sean detectados inequívocamente en las imágenes del par estéreo. Se utilizó la herramienta *tune_blockmatcher* de la biblioteca *StereoVision* con el fin de ajustar los parámetros del detector de objetos y se verificó en paralelo la "calidad" del mapa de disparidad que se muestra en la Figura 7.

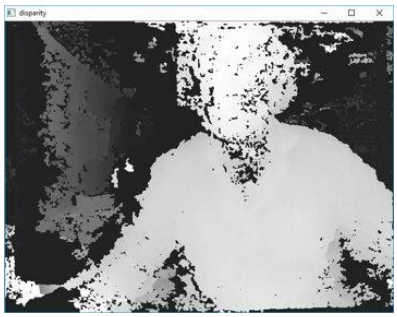

Fig 7. Mapa de disparidad

Los ajustes mencionados se deben realizar en forma iterativa y empírica ya que resulta ser el método más rápido y eficiente. Luego de realizar el ajuste del algoritmo de búsqueda y detección de objetos, se utilizó la función *images_to_pointcloud* (también de la citada biblioteca StereoVision), la que toma como argumentos las matrices de rectificación, un par de imágenes estéreo, los parámetros de ajuste del algoritmo de "*blockmatching*", esto da como resultado una matriz de tres dimensiones que contiene los vectores (también de tres dimensiones) correspondientes a la información del color de cada pixel. Se puede observar la nube de puntos en la Figura 8)

Fig. 8. Resultado de la nube de puntos a partir de la imagen estéreo

V. CONCLUSIONES

Quizás las modificaciones y los agregados realizados sobre el algoritmo DOT no sean suficientes para obtener velocidades considerables en la detección, aunque permiten mejorar adecuadamente la misma. Ahora es aceptable realizar una búsqueda en una imagen HD, pero aun así sigue siendo relativamente lento. Como se plantea al final del apartado "Implementación de DOT en GPU" quizás la respuesta involucre la implementación del proceso de "match template" en la GPU, aprovechando el tiempo ocioso de la misma.

Respecto al algoritmo, presenta fallas que quizás sean mejorables. DOT no considera colores, solo

considera formas, y queda claro que resolver esa falencia sería una mejora a esta implementación. Otro punto importante para destacar es que el tiempo de reconocimiento de un objeto depende de la cantidad de plantillas y de las dimensiones del video. Esto no debería ser así, aunque es una característica intrínseca del algoritmo. Una solución que se podría proponer para la mejora del algoritmo es la de no utilizar la ventana deslizante, sino que por algún medio se detectarán los posibles candidatos a la imagen de la plantilla buscada y luego realizar el "match template" entre las plantillas y esa región de la imagen.

Respecto al cálculo y análisis de profundidad, la calibración de las cámaras y la comprensión del algoritmo están en un nivel muy avanzado, al punto de que solamente sería necesario programar la detección y agregarla a la de objetos. El único inconveniente será el tiempo de procesamiento que se sumaría al que posee la detección DOT y, como se expresó, se debería buscar una solución de compromiso o analizar la unión de ambos algoritmos mucho más detenidamente.

En etapas siguientes de este proyecto de investigación, que sigue vigente en el ámbito de la Universidad, se procederá a continuar el trabajo sobre aquellas cuestiones que hoy se mencionan como dificultades o asuntos pendientes.

VII. REFERENCIAS Y BIBLIOGRAFÍA

A. Referencias bibliográficas:

[1] Bay, Herbert,Tuytelaars, Tinne, Van Gool, Luc.: SURF:Speeded Up Robust Features, European Conference on Computer Vision 2006

[2] Rublee,Rabaud,Konolige,Bradski: ORB: an efficient alternative to SIFT or SURF, IEEE International Conference on Computer Vision 2011

[3] LeCun,Haffner,Bottou,Bengio, Object Recognition with Gradient-Based Learning, CPVR 2004

[4] Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez , Kaiser, Polosukhin, Attention is all you need, Cornell University 2017

[5] Hinterstoisser Stefan, Lepetit Vincent, Ilic Slobodan, Fua Pascal, Navab Nassir.: Dominant Orientation Templates for Real-Time Detection of Texture-Less Objects, Computer Vision Pattern Recognition 2010.

[6] Gonzales, Woods: Digital Image Processing , Tercera Edición, Prentice Hall 2007

[7] Hinterstoisser Stefan, Cagniart Cedric, Ilic Slobodan, Sturm Peter, Navab Nassir, Fua Pascal, Lepetit Vincent.: Gradient Response Maps for Real-Time Detection of Texture-Less Objects, IEEE Transactions on pattern analysis and machine intelligence 2011

[8] OpenCV 9/2/2005 [en línea] Fecha de consulta: 27/7/2020 http://opencv.org/

[9] Martin Montalvo, Técnicas de visión estereoscópica para determinar la estructura tridimensional de la escena, Autor: Curso académico 2009/2010, Máster en Investigación en Informática, Facultad de Informática, Universidad Complutense de Madrid 2009/2010.

[10] StereoVision 15/04/2017 [en línea] Fecha de consulta 27/7/2020

https://pypi.org/project/StereoVision/

Revista Digital del Departamento de
Ingeniería e Investigaciones
Tecnológicas de la Universidad
Nacional de la Matanza
ISSN: 2525-1333. Vol.: 5 - Nro. 1 (AGOSTO-2020)

Recibido: 2020-07-30

Aprobado: 2020-08-06

Hipervínculo Permanente: http://www.reddi.unlam.edu.ar

Datos de edición: Vol. 5-Nro. 1-Art. 8

Fecha de edición: 2020-08-15