ISSN 1870-4069

# Sujeción de objetos virtuales mediante un gesto manual usando Kinect y software libre

Alexis Jair Urquijo-Brito, Abelardo Rodriguez-Leon, Juan Carlos Prince-Avelino, Guillermo E. Ovanco-Chacon, Carlos J. Genis-Triana

[1] Tecnológico Nacional de México/I. T. Veracruz, Veracruz, México

arleon@itver.edu.mx

**Resumen.** El creciente desarrollo de software en el campo de visión por computadora ha sido posible gracias al uso de nuevas herramientas que facilitan la manipulación y procesado de imágenes digitales. Un ejemplo de estas herramientas son librerías como OpenCV y los lenguajes como Processing. En este trabajo se describe el uso de estas herramientas de software libre junto con el dispositivo Microsoft Kinect para detectar el gesto de apertura y cierre de manos. Para lograr la identificación de estos gestos se diseñó, implementó y probó un algoritmo que consta de cinco etapas: adquisición de la imagen, segmentación, detección de contorno, búsqueda de puntos estratégicos y reconocimiento. Se comprueba el correcto funcionamiento del algoritmo en un entorno virtual donde se traslada un rectángulo de un lugar a otro de la pantalla, mediante el gesto de sujetar y soltar de la imagen virtual.

**Palabras clave:** Kinect, OpenCV, Processing, software libre.

# Securing Virtual Objects through a Manual Gesture using Kinect and Free Software

**Abstract.** The growing development of software in the field of computer vision has been possible thanks to the use of new tools that ease the manipulation and processing of images. -an example of these tools are the libraries like OpenCV and languages like Pocessing. In this work describes the use of these free software tools along with the Microsoft Kinect device for the detection of the gesture of opening and closing of hands. To achieve the identification of these gestures an algorithm was designed, implemented and tested, this algorithm is constituted of 5 phases: image acquisition, segmentation, contour detection, search of strategic point and recognition. The correct functioning of the algorithm is verified in a virtual environment where a rectangle is moved from one place to another on the screen, by means of the gesture of holding and releasing the virtual image.

**Keywords:** Kinect, OpenCV, Processing, free software.

## 1. Introducción

El reconocimiento de gestos se ha convertido en un área muy estudiada en la actualidad debido al creciente desarrollo de sus campos de aplicación, que van desde Internet de las cosas, pasando por la salud, robótica, realidad virtual, juegos de computadora, entre otras [1]. Ya que en ocasiones una cámara de color no proporciona información suficiente para analizar una escena, muchas aplicaciones requieren el uso de al menos dos sensores, uno para capturar la información de color y otro para la profundidad (Depth). Este tipo se dispositivos es conocido como cámaras TOF (Time Of Flight), un ejemplo de estos dispositivos es el Kinect de Microsoft [2].

Actualmente existen dos versiones en el mercado para este dispositivo, en este trabajo se utiliza la versión uno lanzada por primera vez en el año 2011. En la búsqueda de la manipulación de objetos virtuales este documento se enfoca en resolver el problema de detectar la apertura y cierre de una mano en cualquier entorno, sin importa el ambiente donde se ejecute, tal gesto es de mucha utilidad para sujetar objetos, arrastrarlos y soltarlos en un lugar determinado La contribución principal de este documentos consiste en desarrollar un algoritmo para reconocer el gesto de apertura y cierre de una mano utilizando algunas fórmulas matemáticas. El propósito de todo lo anterior es lograr identificar un gesto de la mano que simule el tomar un objeto. Dicho gesto inicia con la identificación de la mano abierta. A partir de ese momento se hace un seguimiento de la mano para identificar cuando los dedos se cierran (quedando visualmente dentro de la palma de la mano). Esto se interpreta como el agarre de un objeto virtual (en la pantalla de la computadora). Una vez logrado el agarre virtual este se puede aplicar para la manipulación de objetos en diversos tipos de aplicaciones futuras.

## 2. Trabajos relacionados

Muchos enfoques se han propuesto para buscar reconocer gestos a través de cámaras TOF. Primeramente, se encuentran aquellos que solo usan la cámara de profundidad para realizar el seguimiento y la segmentación de la mano, como propone [5]. La propuesta consiste en un algoritmo que consta de cinco partes: segmentación, extracción de proyección, reducción de error, búsqueda de contorno y aplicación de clasificador basado en redes neuronales. Todo esto usando una antigua y descontinuada librería de software libre llamada OpenNI.

Por otra parte, Dominio [6] propone un algoritmo basado en el reconocimiento de un conjunto de características tridimensionales con la finalidad explotar la información 3D obteniendo la forma y posición de la mano. Específicamente busca explotar cuatro tipos de características: distancia, elevación, curvatura y geometría, para crear un modelo robusto y novedoso usando el Kinect para reconstruir la escena. Posteriormente extrae características descriptivas, dividiendo la mano en palma y dedos. Finalmente propone un clasificador de gestos.

Rem [1] propone el uso de una pulsera en la mano para limitar la forma aplicando el algoritmo de búsqueda RANSAC [1]. Posteriormente la forma de la mano se traslada como una gráfica al dominio de las X donde se busca la posición y tamaño y de cada

dedo. Finalmente propone una nueva métrica de distancia conocida como FEMD, para identificar un gesto de la mano.

Otros modelos como el de Stergiopoulou [7] buscan utilizar redes neuronales para la segmentación y filtrado de color de piel, identificando de manera exitosa las regiones de la mano. Otro enfoque [8] muy utilizado es el uso de mecanismos de detección de articulaciones usando herramientas de software privativo como es el caso de EasyGR que reconoce gestos desde los movimientos del esqueleto usando dos modos de ejecución. El primero modo entrenamiento, donde se almacenan los movimientos en un buffer, este movimiento representa un modelo de gesto que podrá ser reconocido posteriormente.

## 3. Desarrollo

Para el desarrollo del proyecto inicialmente se preparó el entorno de trabajo necesario. Se utilizó Processign [9] como un lenguaje de programación y entorno de desarrollo ya que es software libre y una excelente herramienta para crear prototipos y herramientas profesionales[1]. Se usó también libfreenect [10] como driver y librería para la programación del Kinect. Tambien se usó OpenCV [11] como librería auxiliar en la manipulación de imágenes. Todas estas librerías están bajo licencia GPL.

Posteriormente se vio la necesidad de desarrollar un algoritmo para reconocer el gesto de apertura y cierre de una mano. Se utilizaron fórmulas matemáticas para conseguir una aproximación al centro de la mano. A partir del centro se utiliza la fórmula de la distancia entre dos puntos como métrica de referencia, para saber si todos los dedos se encuentran ya dentro del área de la palma. Este reconocimiento del gesto se hace de manera automática, sin comparaciones con modelos previamente adquiridos, como muchas de las metodologías proponen en la literatura. Estas metodologías comparan las imágenes con modelos previamente almacenados, buscando patrones similares, como es el caso de SP-EMD [2], el cual ha mostrado buenos resultados, aunque se depende en gran medida de los patrones establecidos como referencia.

Como se comentó previamente, para este trabajo se desarrolló un mapeo en dos dimensiones de las imágenes de profundidad y color. La finalidad de este mapeo es eliminar el fondo de la escena tomando ventaja de la cámara de profundidad y estableciendo un umbral valido. Lo anterior nos evita el uso de complejos algoritmos de reconstrucción 3D como los propuestos en otros trabajos de investigación tales como [4] que propone en una comparación cuantitativa de las imágenes. El mapeo de las dos imágenes en una sola, proporciona la ventaja usar solo una imagen 2D, lo que mejora el rendimiento. A diferencia de lo que menciona [5], se ha podido comprobar en este trabajo que la representación 2D si aporta la información necesaria para rastrear la posición de la mano.

Con la finalidad de mejorar la precisión del algoritmo dentro del umbral de decisión, se usó un guante de látex color azul que permitió eliminar cualquier otro elemento en la escena, por ejemplo, el antebrazo. El reconocimiento de colores se procesa haciendo uso del modelo de color HSV (*Hue, Saturation, Value*), comparando el canal H de este

[1] www.processing.org

modelo en busca del color azul, también utilizan algoritmos para la detección de bordes como el Canny Edge Detector [14] y detección de figuras convexas.

Finalmente, se crea un entorno básico virtual de pruebas donde se busca trasladar un rectángulo de un lugar a otro usando gestos de sujeción y procesando el arrastre.

### 3.1. Configuración del entorno

Processing es un proyecto de código abierto que podemos descargar en https://processing.org y está disponible para plataformas Windows, Mac y Linux. Se basa en lenguaje Java y proporciona una colección librerias poderosas y robustas, dentro de un entorno de desarrollo (IDE) propio [12].

Dentro del IDE de Processing se instalaron las librerías necesarias para este proyecto (libfreenect y opencv) de la siguiente manera. En el Menú de herramientas de Processing desplegamos "Sketch", seguido de "Importar biblioteca" y finalmente Añadir biblioteca, esto abre una ventana nueva donde podemos filtrar las bibliotecas por nombre. La biblioteca para el manejo del Kinect se llama "Open Kinect for Processing" y fue escrita por Daniel Shiffman y Thomas Sánchez, antes de proceder con la instalación se recomienda cumplir con los requerimientos. Para un sistema operativo Linux se requiere libusb1.0 y se instala de la siguiente manera.

```
sudo apt-get install libusb-1.0-0 libusb-1.0-0-dev
```

Una vez localizada la biblioteca en el "Contribution Manager" de Processing e instalados los requerimientos procedemos haciendo click en el botón instalar.

De la misma manera buscamos la biblioteca de opencv, la cual se llama "OpenCV for Processing" [13] escrita por Greg Borenstein.

Al finalizar estas instalaciones se obtuvo un entorno funcional para continuar con el proyecto.

### 3.2. Algoritmo de procesamiento

A continuación, se describe el algoritmo para el proceso de detección de gestos. Como se puede observar en la Figura 1 consta de cinco etapas principales, donde cada una de ellas lleva implícitas actividades adicionales. Las etapas son: Adquisición de imagen, segmentación, detección de contorno, búsqueda de puntos estratégicos y reconocimiento. A continuación, se describe con más detalle cada una de ellas.

#### Adquisición de imágenes

Como se ha comentado anteriormente, el Kinect es un tipo de dispositivo TOF, lo que significa que obtendremos dos imágenes diferentes. La imagen de profundidad y la imagen de color o RGB. Ya que ambas cámaras están desfasadas (Fig. 2) es necesaria una calibración previa.

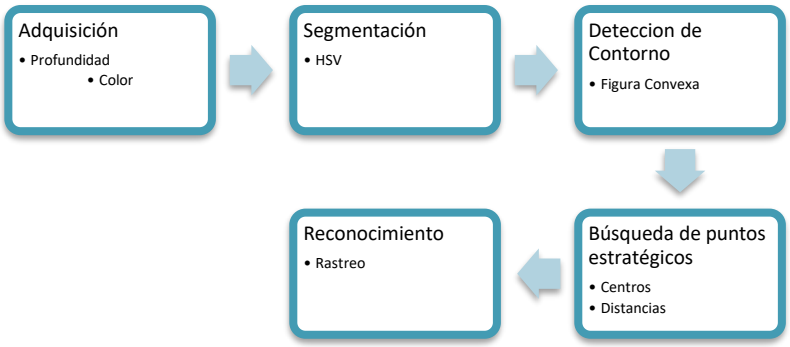


**Fig. 1.** Algoritmo propuesto. Cada etapa consta de actividades complementarias.

**Fig. 2.** Mapeo de imagen de profundidad y color sin calibración.

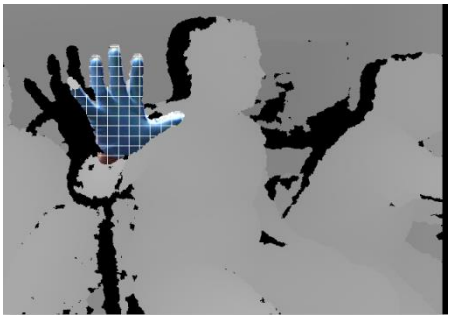

**Fig. 3.** Calibración y mapeo de imágenes de color y profundidad.

Para llevar a cabo la calibración y mapeo, se realizaron los pasos que se observan en la figura 4. Se realizó la calibración de la cámara, moviendo cada pixel de la imagen de color en una matriz temporal, posteriormente los pixeles se desplazan siete por ciento en el eje 'x' y diecisiete por ciento en el eje 'y'. Finalmente para el mapeo se busca un umbral (1.2 metros en este caso) de decisión, donde cada pixel de la cámara de profundidad se compara contra este valor. Cualquier pixel que contenga un valor de profundidad menor al umbral, se sustituye por los pixeles de la imagen obtenida por la cámara de color. (Fig. 3).

```
// Calibración de imágenes
mientras x < 650 //No permite exceder el límite de la imagen
   imagenCalibrada[nuevoX][nuevoY] = imagenColor

// Mapeo de imágenes
mientras i < numeroPixeles
   si(profundidad < rangoVisible) //Distancia de la mano
          nuevaImagen = imagenCalibrada
     si no
      nuevaImagne = imagenProfundidad
```

**Fig. 4.** Pseudocódigo del proceso de Calibración y mapeo

## Segmentación

El siguiente paso es la segmentación de la imagen, que consiste en eliminar todo aquello que no es necesario en la escena, por ejemplo, el antebrazo y el fondo, ya que lo único que nos concierne es la forma de la mano. Para esto se utiliza un guante en color azul (Fig. 5) con la finalidad de limitar el área de la mano. Este guante nos permite enfocarnos en la búsqueda del color en la imagen obtenida. Para ello es necesario transformar la imagen del modelo de color RGB a un modelo HSB, esto permite usar el canal 'H' para hacer un filtrado en la imagen, buscado valores dentro del rango de color azul. Posteriormente a la búsqueda de color, se unifica la imagen eliminando la mayor cantidad de espacios, o huecos. Para lo cual, se usaron algoritmos de erosión y dilatación.

El resultado es una forma clara y limpia de la mano, como la que se muestra en la figura 6.

**Fig. 5.** Guante de látex, utilizado en el rastreo de la mano.

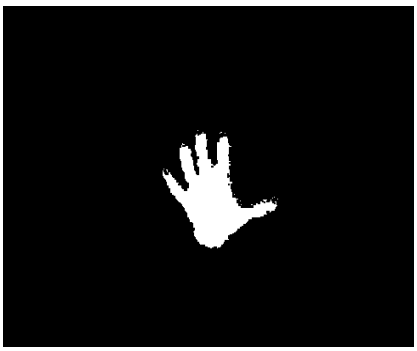

**Fig. 6.** Resultado de la segmentación.

## Detección de contorno en la imagen

Para el análisis del contorno, se utiliza una versión corta del original OpenCV para C++, en el entorno Processing. Aunque esta no es la versión original que contiene cientos de algoritmos para la manipulación de imágenes, fue suficiente para este trabajo, ya que proporciona implementaciones del algoritmo Canny Edge Detector[14]. Un contorno es una lista de puntos que representa una curva en una imagen. Los contornos se pueden manejar de diferentes maneras. OpenCV para Processing los encapsula mediante la clase Contour.

Después de hallar el contorno de la forma (Fig. 7) se reducen el número de puntos, mediante el algoritmo Convex Hull [14]. Lo anterior proporciona un nuevo polígono que representa el contorno de la mano. Convex Hull eliminar las imperfecciones del mapeo de las imágenes, devolviendo una serie de vértices que representan los extremos de todos dedos de la mano y el borde exterior de la palma (Fig. 8).

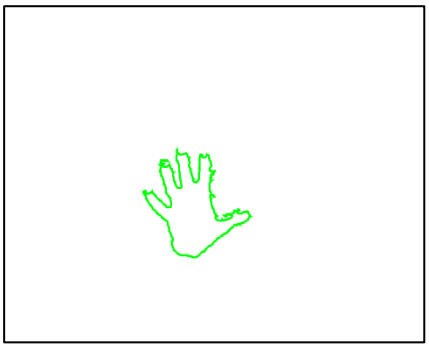

**Fig. 7.** Contorno de la mano.

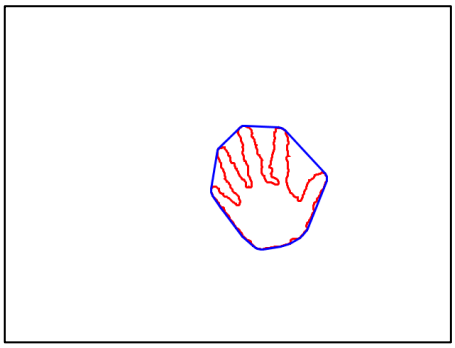

**Fig. 8.** Aplicación de Convex Hull en color azul.

**Búsqueda de puntos estratégicos**

A partir de los vértices que forman el contorno de la mano fue posible encontrar el centro de la misma (Fig. 9) con la siguiente formula:

$$C = \left(\frac{\sum_{i=1}^{len} P[i].x}{len}, \frac{\sum_{i=1}^{len} P[i].y}{len}\right),$$

donde:

‘C’ presenta el centro de la mano,
‘len’ representa el número total de puntos que forman el contorno de la mano,
‘P[i].x’ es las coordenadas ‘x’ de todos los puntos del contorno de la mano,
‘P[i].y’ es las coordenadas ‘y’ de todos los puntos del contorno de la mano.

A partir del centro, se logró delimitar ya la palma de la mano, dibujando un círculo de radio proporcional, a la figura de la mano (Fig. 10), en este caso 120 pixeles.

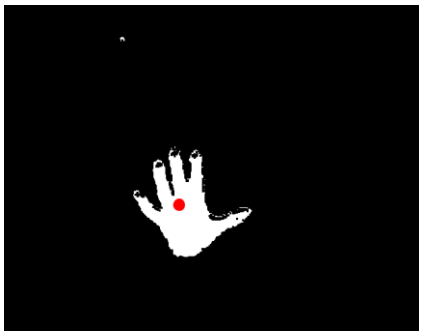

**Fig. 9.** Centro de la mano.

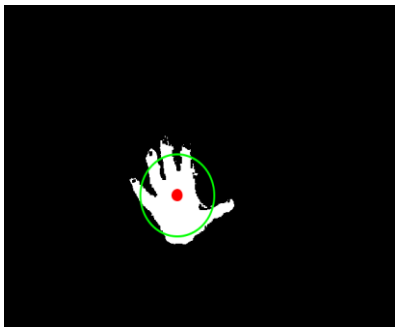

**Fig. 10.** Aproximación al centro de la mano.

Como se mencionó en el punto anterior (detección de contorno de la imagen), la idea era que cada vértice representara un dedo. Esta suposición resulto no ser cierta ya que

en algunas ocasiones aparecían más de un vértice por cada dedo. Estos errores, se deben principalmente a la reducción de la densidad en la imagen, después de la segmentación por color. Para solucionar este problema se aplicó un algoritmo para la reducción de vértices.

El algoritmo para la reducción de vértices, se basó en medir distancias entre todos los vértice e ir comparando sus valores. Las distancias se miden por la fórmula de distancia entre dos puntos:

$$d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}.$$

Este algoritmo considera dos distancias importantes, la distancia menor y la distancia mayor (Fig. 11). La distancia menor se refiere a los puntos en un mismo dedo y la distancia mayor se refiere a la distancia entre diferentes dedos. Cada distancia menor no debe ser de más de 25 píxeles y cada distancia mayor, debe ser superior a 135 píxeles.

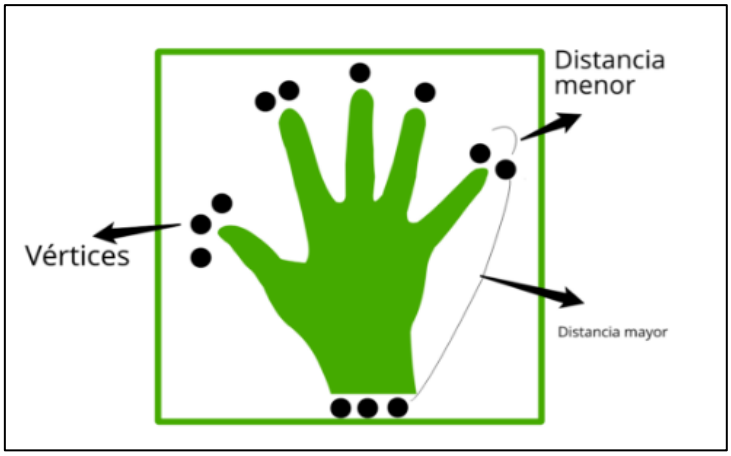


**Fig. 11.** Algoritmo de reducción de vértices.

**Reconocimiento de gestos**

En este punto la detección del gesto se vuelve un proceso más simple. Si se tienen identificados los dedos y la palma de la mano, el reconocimiento del gesto se reduce a determinar cuáles dedos se encuentran dentro del área de la palma de la mano, de tal manera que cuando todos los dedos se encuentran dentro hablamos de una mano cerrada.

## 4. Resultados

Con la finalidad de probar la detección de los gestos, se construyó un entorno virtual, donde se puede apreciar un rectángulo en color azul (Fig. 12), la prueba consiste, en trasladar el rectángulo, de una posición inicial a una posición final, utilizando el gestos de cerrar la mano, para sujetar, y el gesto de abrir la mano, para soltar el objeto.

El experimento anterior se realizó en quince ocasiones, de las cuales, en trece el algoritmo detecto correctamente cuando la mano se cerraba sobre una figura (sujeción). Posteriormente el algoritmo rastreó el recorrido completo de la mano de la mano en las quince ocasiones (incluso en los casos donde no detecto sujeción). Finalmente, el algoritmo detecto la apertura de la mano en las trece ocasiones que si identifico sujeción. Como se muestra en la Tabla 1, el campo "positivos", indica el número de aciertos del algoritmo, el campo "negativos", indica el número de errores.

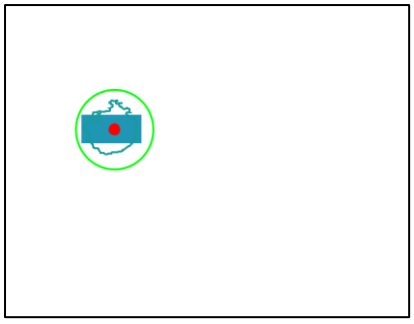

**Fig. 12.** Posición inicial del rectángulo.

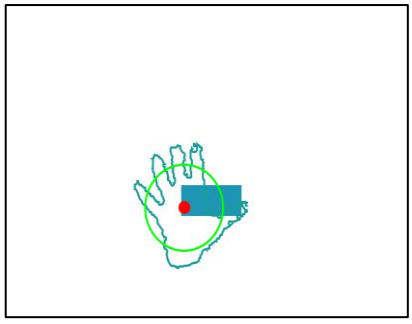

**Fig. 13.** Posición final del rectángulo.

**Tabla 1.** Tabla de resultados.

| Acción | Positivos | Negativos |
|---|---|---|
| Cerrar la mano (sujeción) | 13 | 2 |
| Seguimiento de la mano | 15 | 0 |
| Abrir la mano | 13 | 2 |

## 5. Conclusiones

En este trabajo se diseñó e implemento con éxito un algoritmo de cinco etapas que permite manipular las imágenes de color y profundidad proporcionadas por una cámara TOF al momento de escanear una mano. La finalidad de identificar los gestos de cierre y apertura de la misma se interpretan como sujetar y soltar un objeto. El algoritmo presentado en este trabajo se usará en un futuro en aplicaciones para manipulación de objetos 3D en ambientes virtuales. Específicamente para control de cámara en un simulador 3D desarrollado en el LCI [15] que muestra el comportamiento de un flujo de partículas en un muro de visualización.

Para probar del funcionamiento del algoritmo se usó una pequeña aplicación gráfica donde se debía poder tomar, mover y soltar un pequeño rectángulo mostrado en una pantalla. Los resultados fueron buenos siempre y cuando la mano se encuentre dentro del umbral predefinido en el eje Z. En los casos que el algoritmo no detecto correctamente la sujeción del objeto se debieron a que en esta versión no se consideró la velocidad de cerrado de la mano. Dicha situación se corregirá en un futuro cercano.

## Referencias


1. Ren, Z., Yuan, J., Meng, J., Zhang, Z.: Robust Part-Based Hand Gesture Recognition Using Kinect Sensor. IEEE Transactions on Multimedia, vol. 15, no. 5, pp. 1110–1120 (2013)
2. Herrera C., D., Kannala, J., Heikkila, J.: Joint Depth and Color Camera Calibration with Distortion Correction. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, no. 10, pp. 2058–2064 (2012)
3. Wang, C., Liu, Z., Chan, S.: Superpixel-Based Hand Gesture Recognition with Kinect Depth Camera. IEEE Transactions on Multimedia, vol. 17, no. 1, pp. 29–39 (2015)
4. Villena-Martínez, V., Fuster-Guilló, A., Azorín-López, J., Saval-Calvo, M., Mora-Pascual, J., Garcia-Rodriguez, J., Garcia-Garcia, A.: A Quantitative Comparison of Calibration Methods for RGB-D Sensors Using Different Technologies. Sensors, vol. 17, no. 2, p. 243 (2017)
5. Asad, M., Abhayaratne, C.: Kinect depth stream pre-processing for hand gesture recognition. IEEE International Conference on Image Processing (2013)
6. Dominio, F., Donadeo, M., Zanuttigh, P.: Combining multiple depth-based descriptors for hand gesture recognition. Pattern Recognition Letters, vol. 50, pp. 101–111 (2014)
7. Stergiopoulou, E., Papamarkos, N.: Hand gesture recognition using a neural network shape fitting technique. Engineering Applications of Artificial Intelligence, vol. 22, no. 8, pp. 1141–1158 (2009)
8. Ibañez, R., Soria, A., Teyseyre, A., Campo, M.: Easy gesture recognition for Kinect. Advances in Engineering Software, vol. 76, pp. 171–180 (2014)
9. Processing.org [Online]. Available: https://processing.org. [Accessed: 07- May- 2018].
10. OpenKinect: Openkinect.org. [Online]. Available: https://openkinect.org/wiki/Main_Page. [Accessed: 09- May- 2018].
11. OpenCV library: Opencv.org. [Online]. Available: https://opencv.org. [Accessed: 06- May- 2018].
12. White, S: Processing in Eclipse. Processing.org. [Online]. Available: https://processing.org/tutorials/eclipse/. [Accessed: 09- Jul- 2018].
13. Borenstein, G.: atduskgreg/opencv-processing, GitHub, [Online]. Available: https://github.com/atduskgreg/opencv-processing. [Accessed: 07- Jun- 2018].
14. Kaehler, A., Bradski, G.: Learning OpenCV 3. Sebastopol, CA: O'Reilly Media (2017)
15. Lázaro Velasco, R., Rodriguez-Leon, A., Prince-Avelino, J. C., Ovando-Chacón, G. E., Rodriguez-Gonzalez, A.: Migración de un simulador de partículas 3D al renderizador paralelo Equalizer para su despliegue en un muro de visualización. Revista Coloquio Internacional Multidiciplinario, Vol. 5. no. 1, pp. 778–786 (2017)