Método robusto para detectar dedos usando profundidad

Isaac Juan Rudomín Goldberg, Jorge Adolfo Ramírez Uresti y Christian Jesús Arzate Cruz

ITESM CEM
Atizapan de Zaragoza, Mexico
rudomin.isaac@gmail.com, juresti@gmail.com, arzate.christian@gmail.com

Resumen Se usa una cámara de profundidad de bajo costo para detectar, de manera robusta, manos y dedos. Nuestro método propuesto está basado en filtros de profundidad y análisis de contorno adaptivo. La mayor parte del enfoque propuesto se programó en CUDA. Seleccionamos CUDA para poder conseguir una aplicación de visión computacional en tiempo real. En las pruebas realizadas conseguimos un 95,17 % de efectividad con un 3,2 % de falsos positivos.

Palabras clave: detección, manos, dedos, profundidad, Kinect.

1. Introducción

Sistemas de visión computacional puede ser utilizados para crear interfaces convincentes para computadoras. Esto se debe a que es más natural y divertido controlar una computadora usando el movimiento del cuerpo o gestos de manos y dedos. Estas formas de interfaz también se han visto beneficiadas de los precios bajos que encontramos en varios de los componentes que se usan para desarrollar sistemas de visión computacional. Incluso son lo suficientemente baratos para reemplazar interfaces como botones y palancas.

Muchas interfaces han sido desarrolladas para habilitar interacciones naturales con manos y dedos. La mayoría de estos sistemas de visión computacional usan cámaras RGB tradicionales o infrarrojas [1,?,?,?]. Estos métodos encuentran problemas cuando:

- Las manos y el fondo contienen colores similares
- Cambios de iluminación
- Reflejos
- Sombras
- La geometría del fondo cambia

El objetivo del método presentado en este escrito es: desarrollar un sistema de visión computacional en tiempo real que detecte manos y dedos de forma robusta, utilizando una cámara de profundidad. Escogimos este tipo de cámaras porque simplifican todo el sistema de detección. En mayor parte porque no les

afecta los cambios de iluminación, proveen información geométrica directamente, y la tecnología actual las hace asequibles y precisas.

Parte importante de nuestro sistema es su estructura. Se implementó una plataforma que puede ser utilizada para desarrollar distintos tipos de aplicaciones. Además, la mayoría del los algoritmos fueron programados en CUDA para conseguir que nuestro sistema funcione en tiempo real.

Este artículo continúa con la siguiente estructura. En la siguiente sección se describe el hardware que utilizamos y su posición con respecto al usuario. En la sección III, se hace un análisis sobre las metodologías encontradas en el estado del arte para detectar manos y dedos. La sección IV explica los pasos del proceso que sigue nuestro sistema. Los resultados son presentados en la sección V y las conclusiones y trabajo a futuro aparecen en la sección VI.

2. Hardware y descripción de la Escena

Nuestro sistema consiste de una cámara de profundidad de bajo costo (Microsoft Kinect Sensor) el cual provee un mapa de profundidad a una resolución de 640x480 pixeles a 30 Hz. Objetos entre 0.3 y 3.5 m pueden ser vistos por este sensor. Para mantener nuestro sistema lo más general posible optamos por situar la cámara frente al usuario, de forma que podamos verlo de la cintura a la cabeza. También se considera que cuando el usuario desea interactuar con el sistema, este alza sus brazos y muestra sus palmas y dedos como se puede observar en la Figura 3(a).

3. Estado del arte

En la literatura se pueden encontrar distintas formas de detectar manos y dedos. Por ejemplo, podemos ver en [2,**?**,**?**,**?**] que realizan un análisis de contorno, otros métodos usan algoritmos de reconocimiento de objetos [3], emparejamiento de modelos [4,**?**] o cálculos morfológicos [5] para conseguir su objetivo. Cada método tiene ventajas, desventajas y limitaciones. Ahora analizaremos estos distintos.

3.1. Análisis de contorno

curvatura-k Thiago R. Trigo y Sergio Roberto M. Pellegrino en [1] utilizan el algoritmo curvatura-k para detectar dedos. El primer paso de este método es segmentar las manos y extraer su contorno. Después, para cada uno de los puntos de los contornos encontrados se corre el algoritmo curvatura-k. El algoritmo curvatura-k encuentra el ángulo entre dos vectores (α y β) y toma como entradas; la lista ordenada de puntos que pertenecen al contorno de las manos, y las constantes k y w. La salida de este algoritmo símplemente te dice si el punto en cuestión es parte o no de la curvatura especificada por las constantes k y w. La constante k sirve para formar los ángulos α $(P(i-k))$ y β $(P(i+k)$), como se

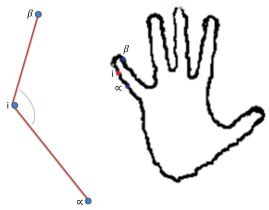


Figura 1. Algoritmo curvatura-k.

puede observar en la Figura 1. La constante w es un umbral medido en radianes. En la aplicación de este artículo los valores de k y w son 20 y 0,95993 radianes, respectivamente. El valor de k se encontró por prueba y error, y el ángulo w al, manualmente, medir los valores de curvatura-k en dedos.

Este método es preciso y rápido pero tiene un gran problema, la definición de las constantes k y w. En especial, encontrar el valor de la constante k es más complicado porque el rango de profundidad (distancia entre la cámara y el usuario) donde un valor específico de k funciona es muy corto.

Aproximación de contorno polígonos En [2] proponen aproximar con polígonos el contorno de las manos, hasta el punto en el que cada dedo se representa con dos vectores. Después a este nuevo contorno aproximado se aplica el algoritmo $ConvexHull$. Los vertices de los polígonos que se ecuentran sobre el contorno encontrado por el algoritmo $ConvexHull$ son considerados como dedos.

El problema que presenta este algoritmo es similar al anterior, se tiene que calibrar para cada aplicación en especial. Esto se deba a que la aproximación con polígonos debe de ser ajustada dependiando de la distancia entre la cámara y el usuario. Se realizaron pruebas y no se consiguió obtener un nivel de aproximación que funcionara para el rango de visión de nuestro sensor de profundidad.

3.2. Algoritmos de reconocimiento de objetos

Estos métodos usan algoritmos supervisados de aprendizaje, por lo tanto, se necesita entrenar el sistema antes de poder usarlo. Adaboost [6], es el algoritmo más popular para esta tarea porque es rápido y preciso. Esto se debe a que usa clasificadores débiles para determinar si una imagen contiene o no el objeto de interés. Aunque este método es preciso, es muy difícil conseguir una buena base de datos de clasificación por todo el trabajo que se requiere y algunos problemas a los que puede ser susceptible, como el sobre-entrenamiento. Tampoco es tan rápido como al algoritmo curvatura-k.

3.3. Emparejamiento de modelos

La idea básica del emparejamiento de modelos es crear el modelo de un objeto de interés, en este caso manos y dedos, y luego buscar en la imagen

que se quiera objetos que se emparejen con el modelo. El modelo puede ser creado usando emparejamiento de bordes, emparejamiento en escala de grises o emparejamiento de gradiente. Para atacar el problema de la rotación y variación de escala, este método necesita buscar usando diferentes modelos. Para un rango específico es preciso pero el buscar en una imagen el modelo sin importar la escala o rotación, reduce sustancialmente el desempeño de este algoritmo.

3.4. Cálculos morfológicos

La premisa de este método es que los dedos son largos y delgados en comparación con la mano. Por lo tanto, desaparecen cuando un operador de erosión es aplicado a una imagen que contenga una mano. El operador de abrir puede ser utilizado para detectar dedos al escoger una ventana apropiada y luego detectando la resta con la imagen original. El resultado de la resta son blobs de dedos. Como sucede con los métodos previamente analizados, este algoritmo necesita ser ajustado para que funcione en una aplicación en específico.

4. Nuestro sistema

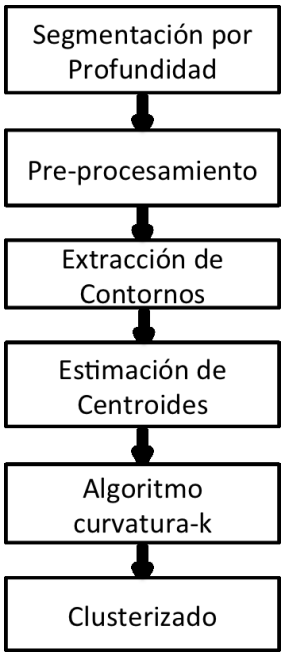
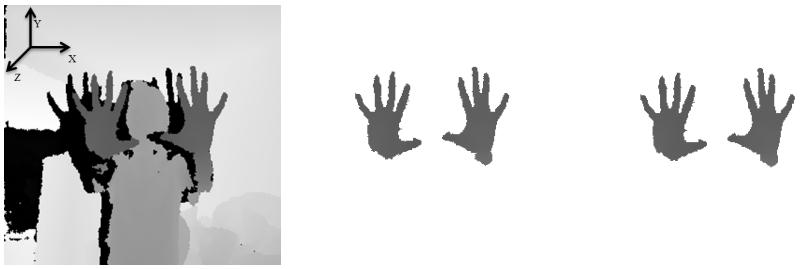


Figura 2. Diagrama de flujo del algoritmo de setección de Dedos.



(a) *Izquierda*: Mapa de Profundidad Original. *Centro*: Segmentación por Profundidad. *Derecha*: Pre-procesamiento

(b) *Izquierda*: Extracción de Contornos. *Centro*: curvatura-k. *Derecha*: Centroides de las Manos y Detección de Dedos

Figura 3. Resultados de cada uno de los pasos de nuestro algoritmo.

La Figura 2 muestra los pasos que nuestro método sigue, también podemos observar la imagen resultante de cada paso. En las siguientes sub-secciones se explicará cada paso en detalle:

1. *Segmentación por Profundidad:* Asumiendo que el objeto más cercano a la cámara es el objeto de interés, manos, usamos un filtro de profundidad. Este filtro elimina de la escena todos los pixeles que están fuera del rango $Rango(PC-l)$, dónde PC es el valor, en cm, del píxel más cercano a la cámara. De manera experimental se encontró el valor óptimo de l que funciona en el rango de nuestra cámara de profundidad. Este valor óptimo encontrado fue de $35cm$ y en la Figura 3(a) podemos ver la imagen que obtenemos al aplicar este paso.
2. *Pre-procesamiento:* Primero, un filtro Guassiano es usado para reducir el ruido de la cámara y asegurarnos de que pixeles aislados no formen parte de nuestro objeto de interés. Luego un filtro de cerrado morfológico es aplicado para llenar huecos y suavizar los orillas externas (Figura 3(a)).
3. *Extracción de Contornos:* Los contornos son extraídos usando la implementación de OpenCV de *Approximate Freeman Chains* [7] (Figura 3(a)). Para los pasos siguientes solamente vamos a tomar en cuenta aquellos contornos

que sean más grandes que un umbral dado. Este umbral se encuentra de forma experimental y funciona en el rango de 0.3 a 3.5 m.

4. *Estimación de Centroides:* Para distintas aplicaciones es útil conocer el centroide de las manos (Figura 3(b)) para seguir su posición. Para obtener su centroide, es fácil el calcular los momentos Hu [8] para cada uno de los contornos con las siguientes fórmulas:

$$A = m_{0,0}$$
$$x_c = \frac{m_{1,0}}{A} = \frac{m_{1,0}}{m_{0,0}}$$
$$y_c = \frac{m_{0,1}}{A} = \frac{m_{0,1}}{m_{0,0}}$$

dónde (xc, yc) es el centro de gravedad del objeto

5. *Algoritmo curvatura-k:* Por su velocidad y precisión es que seleccionamos este algoritmo para detectar dedos. Su única desventaja es encontrar el valor adecuado de la constante k. Para resolver este problema se definió una constante k general que funciona en el rango de 0,3 a 3,5m que nuestro sensor de profundidad maneja. Usando nuestro valor k general (15) conseguimos entre 10 y 20 puntos de interés por dedo. Podemos ver el resultado en la Figura 3(b).
6. *Clusterizado:* Para conseguir la posición 3D de cada dedo es necesario clusterizar los puntos obtenidos en el paso anterior. Para conseguir nuestro objetivo y mantener la velocidad del algoritmo de detección de dedos, se implementó una versión modificada de vecinos cercanos para realizar esta tarea. La idea básica de este método es usar el algoritmo de vecinos cercanos para centrar los centros que el algoritmo AS compute, este proceso se repite hasta que todos los puntos estén clasificados o se llegue a un máximo de iteraciones. En las siguientes líneas se muestra el pseudocódigo de este algoritmo:

```
// Repetir hasta que se llega al umbral
while continuar
      // Computar el algoritmo de vecinos cercanos
      CentrosInfo = vecCer(centros, inf, umbralK)

      // Computar el algoritmo AS
      [centrosNuevos] = as(centrosInfo, umbralAS)

      // Revisar si se tiene nuevos centros
      if centrosNuevos == centros
        continuar = 0
        break
      end
end
```

5. Resultados

Se le pidió a 4 personas que probaran nuestro sistema y conseguimos un 95,17 % de efectividad con un 3,2 % de falsos positivos en 200 cuadros que fueron analizados. Es difícil comparar directamente el desempeño de nuestro método con otros porque no hay un estándar para evaluar este tipo de aplicaciones. Pero podemos concluir que:

- Es más robusto que métodos que usan cámaras RGB
- Es más robusto que métodos que necesitan calibrarse para cada aplicación

6. Conclusiones y trabajo a futuro

Nuestro objetivo se cumplió. Este artículo presenta un sistema de detección de manos y dedos que usa mapas de profundidad y análisis de contorno adaptivo. Nuestra técnica puede detectar manos y dedos en todo el rango en el que nuestro sensor funciona. Fue implementado en CUDA la mayor parte de los algoritmos, con lo que conseguimos una aplicación de visión computacional en tiempo real. Tampoco es afectado por cambios de iluminación, reflejos o sombras. Como trabajo a futuro desarrollaremos aplicaciones que hagan uso del sistema propuesto en este artículo. Principalmente nos interesa trabajar en interfaces naturales que detecten gestos de manos y dedos en 3D.

Referencias

1. Trigo, T.R., Pellegrino, S.R.M.: An analysis of features for hand-gesture classification. 17th International Conference on Systems, Signals and Image Processing (2010)
2. L.C. Ebert, G. Hatch, M.T.S.R.: Invisible touch—control of a dicom viewer with finger gestures using the kinect depth camera. Journal of Forensic Radiology and Imaging, Volume 1, Issue 1 (2013)
3. Yun, L., Peng, Z.: An automatic hand gesture recognition system based on viola-jones method and svms. 2009 Second International Workshop on Computer Science and Engineering (2009)
4. H. Koike, Y. Sato, Y.K.H.T., Kobayash, M.: Visual tracking of bare fngers for interactive surface. ACM Symposium on User interface Software and Technology (UIST '04) (2004)
5. Dung Duc Nguyen, T.C.P., Jeon, J.W.: Fingertip detection with morphology and geometric calculation. The 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (2009)
6. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences (1997)
7. Seo, N.: Tutorial: Opencv haartraining (rapid object detection with a cascade of boosted classifiers based on haar-like features). http://note.sonots.com/SciSoftware/haartraining.html (2008)
8. Hu, M.K.: Visual pattern recognition by moment invariants. IRE Trans. Inform. Theory (1962)