# INSTITUTO TECNOLÓGICO DE COSTA RICA

## PRIMER PROYECTO PROGRAMADO

*Ariel Herrera*
*Jorge Sibaja*
*Saúl Zamora*

profesor
M. Sc. Saúl Calderón Ramírez

August 19, 2016

## I. Introducción

El análisis de contenido de video o en inglés *Video Content Analysis, Video Content Analytics o VCA* es la habilidad de analizar autimáticamente un video para determinar eventos temporales y/o espaciales.

Esta capacidad técnica es usada en un amplio rango de dominios, los cuales incluyen entretenimiento, cuidados de la salud, ventas, automotores, transporte, automatización de hogares, detección de fuego y humo, seguridad, entre otros. Los algoritmos pueden ser implementados por software en máquinas de propósito general o en hardware especializado en procesar unidades de video.

Muchas funcionalidades distintas pueden ser implementadas en VCA. Detección de movimiento en video es una de las formas más simples en las que el movimiento es detectado respecto a una escena de fondo inmóvil.

VCA depende de video de buena calidad como entrada, así que usualmente es combinado con tecnologías de mejoramiento de video como super resolución, estabilización de imagen, entre otros.

## II. Análisis del problema

Actualmente la Universidad de Costa Rica desarrolla un sistema de análisis de video con el fin de analizar automáticamente videos de futbol. Las etapas de éste van desde la identificación de escenas con información útil en el video, segmentación y rastreo de jugadores, hasta el análisis automático de los recorridos y comportamiento de los jugadores.

El presente proyecto se enfoca en el desarrollo de la etapa de segmentación de jugadores.

## III. Diseño de la solución

El algoritmo implementado utiliza la información conocida con anterioridad, basada en el hecho de que los jugadores siempre se encuentran rodeados de pixeles verdes (el campo de juego). Dado esto, el algoritmo se divide en dos etapas:

- Detección del campo de juego o *Máscara de cancha*
- Detección de los jugadores o *Máscara de jugadores*

Para obtener el área correspondiente a los jugadores, se realiza una operación AND entre las máscaras.

### A. Máscara de la cancha

Para obtener el campo de juego, se sigue el siguiente procedimiento:

1) Se convierte la imagen de entrada a una imagen de cromaticidad *H*, tomando la capa de cromaticidad H.
2) Se definen los valores que determinan el rango de verdosidad, para umbralizar la imagen dentro de ese rango y obtener la máscara de pixeles verdosos.
3) Se rellenan los hoyos de la máscara de verdosidad que podrían corresponder a los jugadores.
4) Eliminar las regiones "adulteradas" pequeñas (menos del 10% de la imagen).
5) Eliminar la parte correspondiente al marcador del juego.

### B. Máscara de los jugadores

Para obtener los jugadores, lo que se hace es:

1) Se convierte la imagen de entrada a una imagen de cromaticidad *H*, tomando la capa de cromaticidad H.
2) Se normaliza la imagen en un rango de 0 a 255.
3) Se calcula la varianza local usando como entrada la imagen de cromaticidad *H*
4) Para calcular el umbral óptimo para la imagen, se utilizó la funcion *graythresh* del lenguaje Octave, la cual por defecto usa el algoritmo de *Otsu*.

### C. Pseudocódigo

```
funcion procVideo(dirVideo)
        infoVideo = obtInfo(dirVideo);
        for (i = 1; i < infoVideo.cantFrames; i++)
                frameHSV = obtFrameHSV(dirVideo, i);
                mascCancha = obtenerCancha(frameHSV);
                mascJugadores = obtenerCandidatos(frameHSV);
                mascFinal = mascCancha & mascJugadores;
                guardarMasc(mascFinal, nombre);

funcion obtFrameHSV (dirVideo, i)
        frame = obtFrameVideo(dirVideo, i);
        hsvFrame = convRGBHSV(frame);
        return hsvFrame;

funcion obtenerCancha(frameHSV)
        mascVerde = (obtener Mascara de verde);
        rellenar = rellenarAgujeros(mascVerde);
        Ceros = elimPorcionesMenores(rellenar);
        complemento = complementar(Ceros);
        Unos = elimPorcionesMenores(complemento);
        mascCancha = complementar(Unos);
        return mascCancha;

funcion obtenerCandidatos(frameHSV)
        tamFrame = tamanno(frameHSV);
        frameHSV = frameHSV.*255;
        matrizVarianzas = ceros(tamFrame);
        recorrerMatrizH:
                ventana = slice(frameHSV);
                matrizVarianzas(i, j) = calcVarianza(ventana);
        frameHSV = frameHSV.^(0.5)
        frameHSV = frameHSV./255
        bordes = varianzaABin(frameHSV, calcUmbral(frameHSV));
        mascJugadores = funcionRellenarAgujeros(bordes);
        return mascJugador;
```

## IV. Pruebas

Fig. 1. Máscara de la cancha en el frame 1

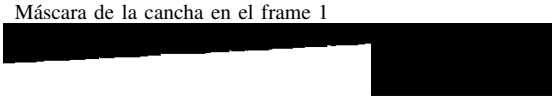


Fig. 2. Máscara de la cancha en el frame 90

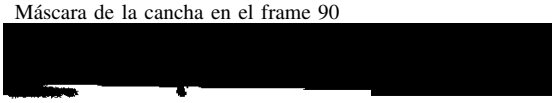


Fig. 3. Máscara de la varianza en el frame 1

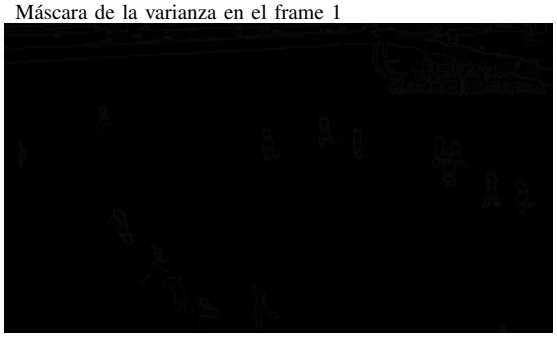


Fig. 4. Máscara de la varianza en el frame 90

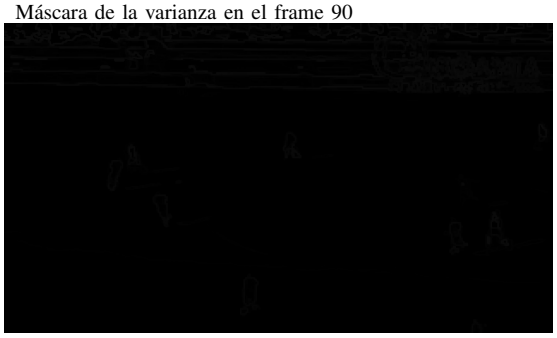


Fig. 5. Resultado final en el frame 1

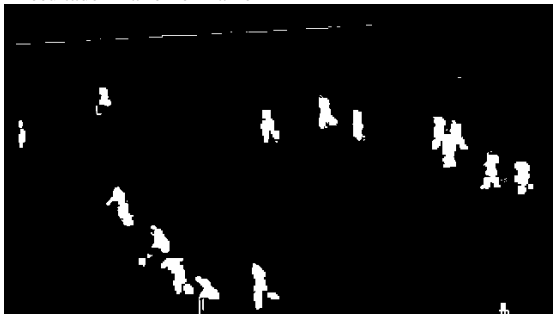
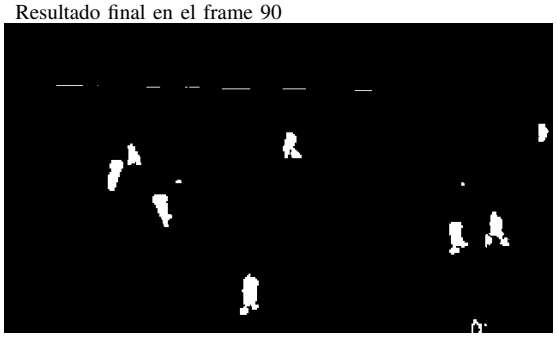


Fig. 6. Resultado final en el frame 90



## V. Conclusiones

- La forma en la que esta diseñado el lenguaje *Octave* lo hace muy práctico para el manejo de señales.
- Dado que las matrices son un tipo de dato nativo del lenguaje, se facilita mucho el manejo y manipulación de imagenes, ya que estas se convierten en matrices numéricas.
- Otro aspecto que se facilita por las matrices como tipo de dato primitivo, es la automatización de recorridos de estructuras de datos. Dado que las matrices son tipos de datos primitivos, se elimina la necesidad de ciclos y/o ciclos anidados *(fors dentro de fors o whiles)* para recorrer las matrices.
- El lenguaje presenta una manera muy práctica y fácil de agregar las librerías necesarias.

## VI. Referencias

## References

[1] Community, T. O.-F. (2009, June 7). *Documentation*. Retrieved August 17, 2016, from `http://octave.sourceforge.net/docs.html`

[2] W, J. (1996). *GNU octave*. Retrieved August 17, 2016, from `https://www.gnu.org/software/octave/doc/v4.0.1/`

[3] *Octave - preface*. Retrieved August 17, 2016, from `http://www.math.utah.edu/docs/info/octave_1.html#SEC27`