DOCUMENTO DE PRUEBA DE CONCEPTO

Valeria Calderón, Lucy Chaves, Seth Stalley

20 de agosto de 2016

Este documento es parte de la implementación mínima del proyecto Sport Analytics, realizada para probar la factibilidad de varias tecnologías y algoritmos propuestos para la detección y clasificación automática de jugadores en videos de fútbol.

Índice

1. Metodología Scrum

Para seguir la metodología Scrum en Sport Analytics, se creó un proyecto homónimo en la herramienta libre Zoho Projects en el portal acs2016[1]. No se presentaron problemas mayores a la hora de crear el proyecto o de incluir a los miembros del equipo de trabajo. La herramienta parece ser adecuada para la

[1] https://projects.zoho.com/portal/acs2016#dashboard/964323000000015005

metodología Scrum, por su completitud en cuanto al manejo de tareas y reportes se trata.

2. Herramienta de control de versiones

El control de versiones se llevó acabo integrando el entorno de desarrollo con Github. Para esto, se creó el repositorio IC6831-Project1[2]. El equipo de trabajo no experimentó problemas con la herramienta, debido a que es normalmente usada por sus miembros.

En el repositorio se encuentra tanto el desarrollo de la aplicación en Java, como la implementación inicial de la interfaz en AngularJs. En el repositorio se trabajó con una serie de branches, donde se tiene un branch «master» que es la versión estable del programa, un branch de «QA» para realizar pruebas antes de hacer un pull request a master, entre otros. Siguiendo esta metodología de trabajo cada «feature» debe ser trabajado sobre su propio brach, subido a QA y finalmente a master, una vez que se alcance la versión aceptable.

3. Diagrama de componentes

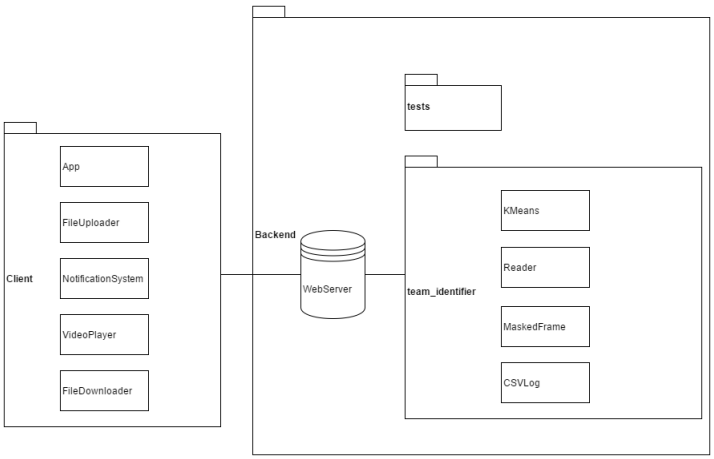

Fig 1. Diagrama de componentes del sistema a desarrollar

4. Análisis de trozos del sistema

4.1. Algoritmos basados en OpenCV (Java)

Los trozos implementados y su respectivo análisis se incluyen a continuación:

[2]https://github.com/SethStalley/IC6831-Project1

4.1.1. Trozo 1

El primer trozo consiste en la lectura del video y la obtención de los frames a partir de un url de prueba. Para esto se obtuvo el número total de frames en el video proporcionado, cabe mencionar que en este aspecto se tuvo problemas, ya que Java no reconocía la constante CV_CAP_PROP_FRAME_COUNT.

Se encontró que existe un bug relacionado ya reportado y se procedió a declararla como una constante de la clase según los valores especificados en la pregunta OpenCV Constants.CaptureProperty[3] de Stack OverFlow.

```
public void readVideo(String url) throws FileNotFoundException{
    System.loadLibrary(Core.NATIVE_LIBRARY_NAME);

    VideoCapture cap = new VideoCapture();
    cap.open(url);

    int numFrames = (int) cap.get(CV_CAP_PROP_FRAME_COUNT);

    for(int i = 1; i < numFrames; i++){
        Mat frame = new Mat();

        cap.read(frame);
        frames.add(frame);
        }

    convertToHsv();
}
```

Fig 2. Lectura del video y obtención de los frames

4.1.2. Trozo 2

Posterior a la lectura del video y obtención de los frames, se convirtió cada uno de estos al modelo de color HSV haciendo uso de la función cvtColor y la especificación COLOR_RGB2HSV. Cada uno de los frames resultantes se agregó a una lista, pero únicamente incluyendo el canal H para evitar que los frames se viesen afectados por sombras, luces u otros factores del entorno. Durante la implementación de este trozo no se presentaron problemas.

```
public Mat getHueChannel(Mat hsv){
    List<Mat> channels = new Vector<Mat>();
    Core.split(hsv, channels);

    return channels.get(0);
}

public void convertToHsv(){
    for(Mat frame:frames){
        Mat framehsv = new Mat(frame.height(), frame.width(), frame.type());
        Imgproc.cvtColor(frame, framehsv, Imgproc.COLOR_RGB2HSV);
        framesHsv.add(getHueChannel(framehsv));
    }
}
```

Fig 3. Conversión al modelo HSV y extracción del canal H

[3]http://stackoverflow.com/questions/21066875/opencv-constants-captureproperty/21066952#21066952

4.1.3. Trozo 3

Luego de obtener el canal H de cada uno de los frames, se procedió a obtener una máscara a partir de un rango de verdosidad de la cancha. Pese a ser pocas líneas de código, la implementación de este trozo requirió más tiempo, ya que la obtención de valores adecuados únicamente fue posible mediante prueba y error.

```
public static Mat getRange(Mat image){
    Mat mask = new Mat();

    Scalar lowerGreen = new Scalar(35, 50, 50);
    Scalar upperGreen = new Scalar(70, 255, 255);
    Core.inRange(image, lowerGreen, upperGreen, mask);

    return mask;
}
```

Fig 4. Obtención del rango de verdosidad de la cancha

4.1.4. Trozo 4

Se probó la transformación morfológica «Dilatación», para conocer los efectos que tenía en la imagen como reemplazo a la función imfill de MATLAB.

```
public static Mat dilate(Mat hsv){
    int dilation_size = 2;
    Mat result = new Mat(hsv.rows(), hsv.cols(), hsv.type());

    Mat kernel = Imgproc.getStructuringElement(Imgproc.MORPH_RECT, new Size(2*dilation_size + 1, 2*dilation_size+1));
    Imgproc.dilate(hsv, result, kernel);

    return result;

}
```

Fig 5. Dilatación aplicada a una imagen HSV.

4.1.5. Otros trozos

Además de los trozos anteriores, se probó la normalización de imágenes HSV y el cálculo de la varianza local para la detección de regiones candidatas a ser jugadores.

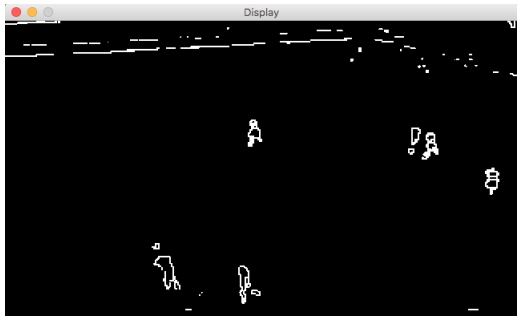

Fig 5. Primeros resultados obtenidos

En general, se tuvo problemas a la hora de encontrar documentación adecuada para la biblioteca OpenCV + Java. La mayoría de tutoriales y guías están desarrolladas para C++ y Python, de hecho, en la mayoría de casos es mucho más sencillo implementar los algoritmos en esos lenguajes que en Java, por lo que quizás sería recomendable analizar la escogencia del lenguaje cuando se desarrollen proyectos de este tipo.

4.2. Implementación inicial del algoritmo K-medias

La implementación inicial del algoritmo K-medias consiste en el cálculo de la distancia de Bhattacharyya. El reto mas grande aquí fue convertir el algoritmo, expresado como uno puramente matemático, a un algoritmo computacional. El trozo desarrollado es el siguiente:

```
public class KMeans {
    //Number of clusters, we default to 2 because there are two soccer teams.
    private int NUM_CLUSTERS = 2;

    public KMeans(int numClusters) {
        this.NUM_CLUSTERS = numClusters;
    }


    /**
     * // Calculate Bhattacharyya distance.
     * @param X:Closely grouped Data
     * @param U: Means Sample Data
     * @return - double value.
     * @throws Exception
     */
    private static double distance(HSV X[], HSV U[]) throws Exception {
        double sum = 0.0;

        if(X.length != U.length) {
            throw new Exception("Muestras no son del mismo largo!");
        }

        for(int i = 0; i< X.length; i++) {
            sum += Math.sqrt(Math.abs(X[i].value - U[i].value));
        }

        return sum;
    }

    public class HSV {
        double value;
    }

}
```

Fig 7. Cálculo de la distancia de Bhattacharyya

4.3. Implementación inicial de la interfaz web usando AngularJs 2.0.

Se implementó una versión inicial de la interfaz web. Se contruyó un método fácil para agregar y subir a varios archivos (videos) para ser procesados por el backend. Ademas, se agregó un sistema preliminar de notificaciones para poder enviar mensajes al usuario mientras interactúan con la interfaz. Finalmente, se implementó un demo donde se reproduce un video almacenado en el backend por medio de HTML5.

Se presentaron dificultades por el hecho de utilizar Java como servidor http, principalmente por el uso de Anuglar 2.0 junto con un backend de Java. No es posible simplemente servir los archivos estáticos de Angular 2.0, ya que este usa typscript para sus componentes, lo cual debe ser compilado a js para producción. Al final, se logró un flujo de desarrollo aceptable, que consiste en trabajar sobre la interfaz usando un servidor light de Node, una vez que se han realizado los cambios deseados al front-end, se compila usando Angular con Node y se guardan lo archivos generados en un camino destinado para archivos estáticos dentro de la configuración del backend (Spring).

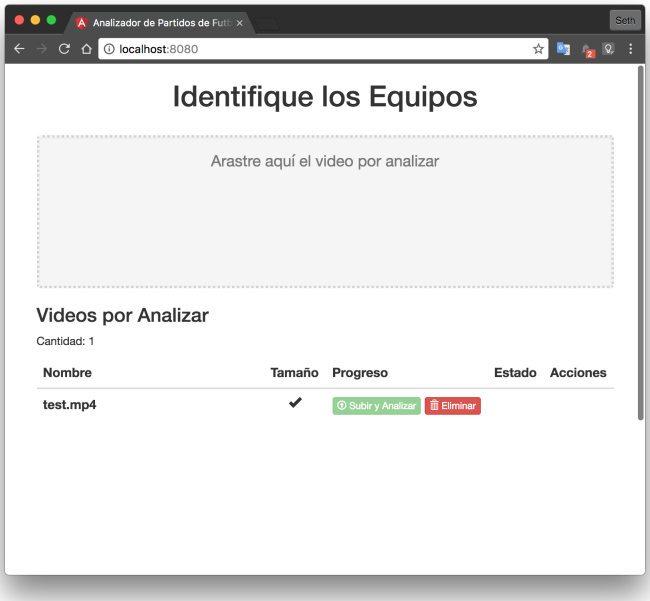

Fig 8. Pantalla inicial para carga del video

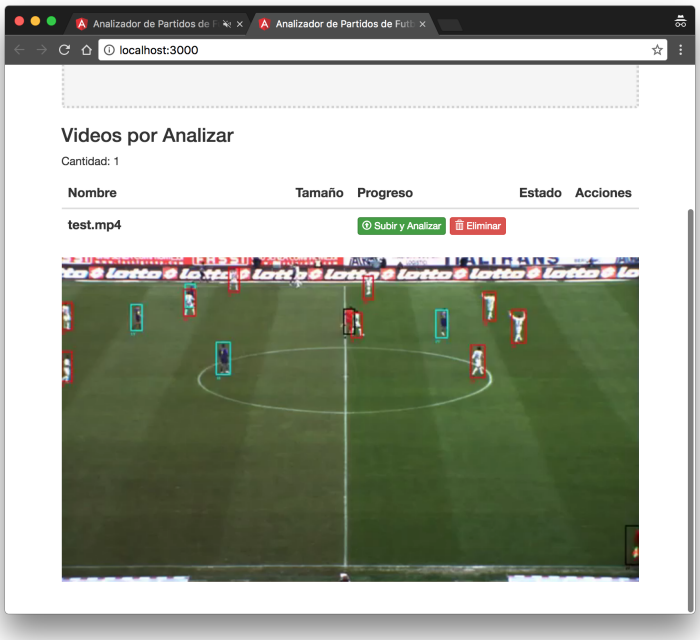

Fig 9. Pantalla de resultado

4.4. Pruebas Unitarias con JUnit

Se creó un paquete llamado «tests» que contiene todas las clases con las unidades de pruebas especificadas. Se hicieron algunos métodos de prueba «dummy» para aprender sobre las funcionalidad y forma de trabajar de JUnit, los cuales deben ser corrido siempre antes de hacer commit de alguna nueva funcionalidad al repositorio.

5. Tiempos de desarrollo

La estimación de tiempos de desarrollo de los módulos del sistema se resumen en el siguiente cuadro:

Módulo	Tiempo estimado (Hrs)
Client	45
Web Server	30
Team_Identifier	150
Tests	3