Prueba de Concepto

Sistema de análisis deportivo

Instituto Tecnológico de Costa Rica
Escuela de computación
Ingeniería en computación
Aseguramiento de la calidad de software

Joshua Mata Araya
Adrian López Quesada
Josué Arrieta Salas

21 de agosto de 2016

Índice

1. Diagrama de componentes **2**

2. Tiempos de desarrollo estimados **2**

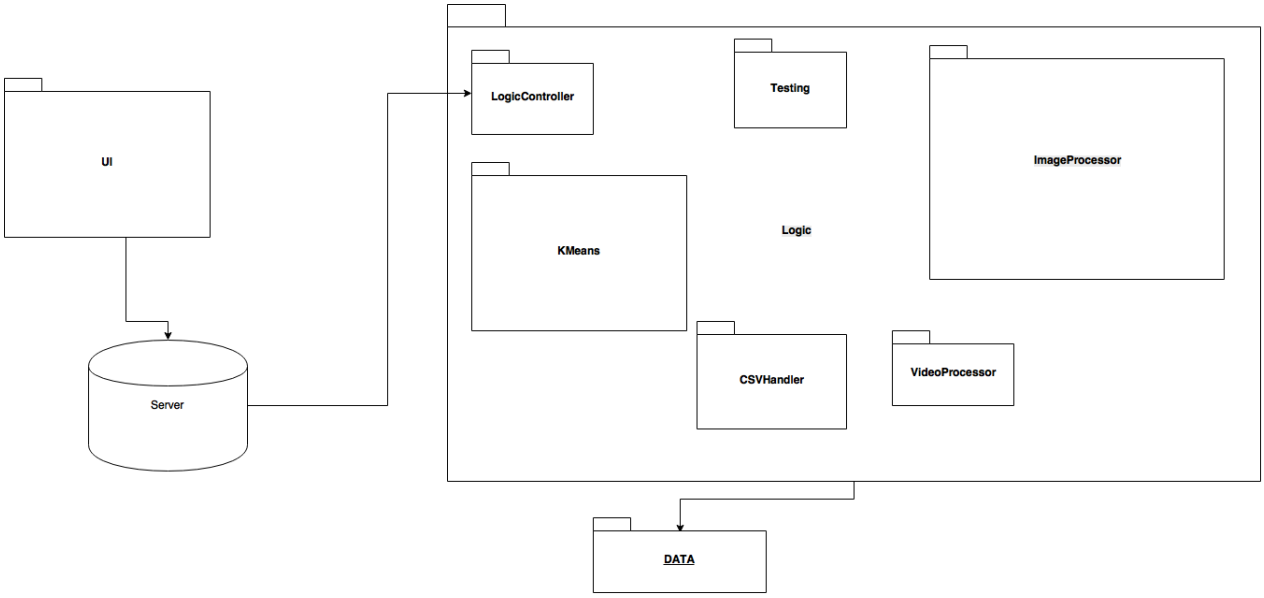
3. Análisis básico de trozos implementados **2**
3.1. greenMask() - opencv 2
3.2. imfill() - opencv 3
3.3. bwareopen() - opencv 4
3.4. detectField() - opencv 5

4. Detalles de problemas encontrados con las herramientas **6**
4.1. Problemas en desarrollo 6
4.2. Problemas en organización del equipo de trabajo 7
4.3. Problemas en concepto del software 7

5. Github **7**

6. Gestion del proyecto **7**

1. Diagrama de componentes



NOTA: Las clases del componente "Logic.en este momento están públicas para que puedan ser visibles dentro del paquete "Testing", sin embargo esto será analizado con mayor detalle.

2. Tiempos de desarrollo estimados

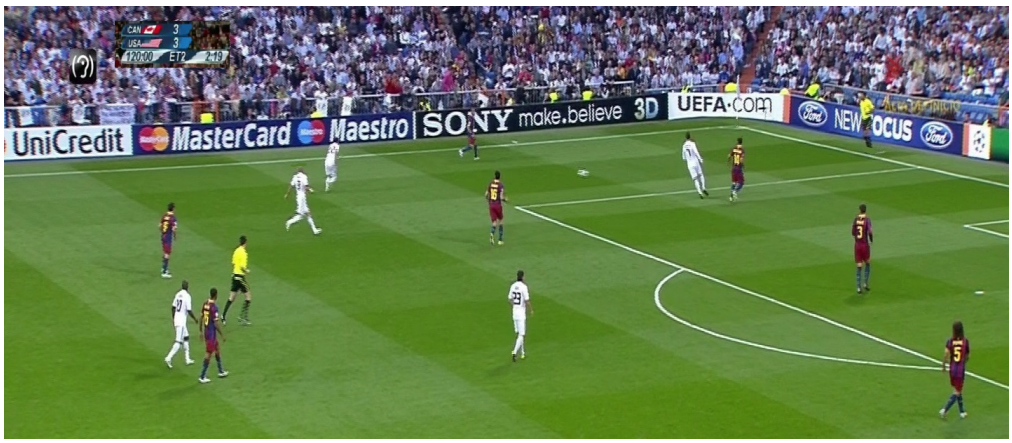
Módulo del sistema	Tiempo de desarrollo estimado
UI	20 Hrs
Server	20 Hrs
LogicController	6 Hrs
KMeans	12 Hrs
CSVHandler	4 Hrs
VideoProcessor	4 Hrs
ImageProcessor	4 Hrs

3. Análisis básico de trozos implementados

En esta sección se presentarán los algoritmos considerados más importantes que fueron implementados en este primer Sprint. Se hará un pequeño análisis acerca de su funcionamiento; además de las entradas y salidas de los métodos. Se considera código tanto del Frontend como del Backend.

3.1. greenMask() - opencv

Este método recibe una imagen (en formato hsv), y devuelve otra imagen con una máscara binaria de pixeles "verdosos". Se tuve que definir un umbral de "verdosidad"que va desde 37 a 63 (para la capa H). Para las capas S y V se escoge un umbral de 100 a 255. En caso de querer cambiar este umbral, simplemente se modifican las constantes. Particularmente en este método no hubo ningún problema a la hora de la implementación debido a la mucha documentación que había, y a una comunidad activa y grande de opencv. Se tiene a siguiente imagen original:



Se produce la siguiente imagen binaria:

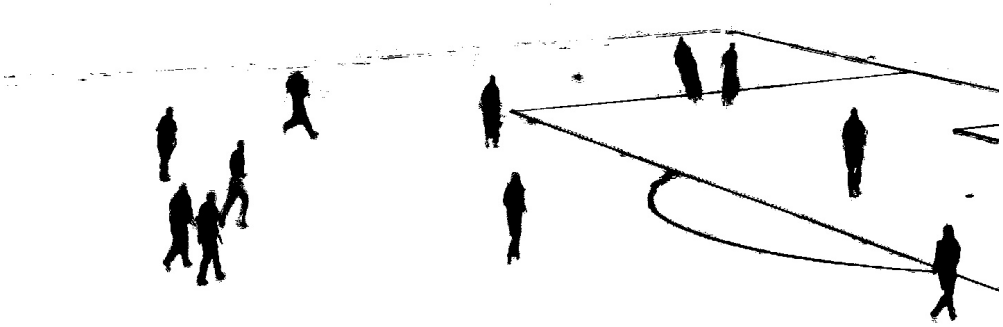
3.2. imfill() - opencv

Tiene la misma funcionalidad que la función imfill en Matlab. Su entrada es una imagen binaria, y su salida es la misma imagen binaria, pero con los hoyos rellenados. Al detectar el campo se utiliza para rellenar hoys que corresponden a posibles jugadores.

```
public static Mat imfill(Mat pImage){
        Mat clone = pImage.clone();
        Mat mask = new Mat(clone.rows() + 2, clone.cols() + 2, CvType.CV_8UC1);
        Imgproc.floodFill(clone, mask, new Point(0,0), Constants.WHITE);
        /*starts to fill in point (0,0)*/

        Mat invertedImage = new Mat();
        Core.bitwise_not(clone, invertedImage);//image complement

        Mat filledImage = new Mat();
        Core.bitwise_or(pImage, invertedImage, filledImage);//or between images

        return filledImage;
    }
```

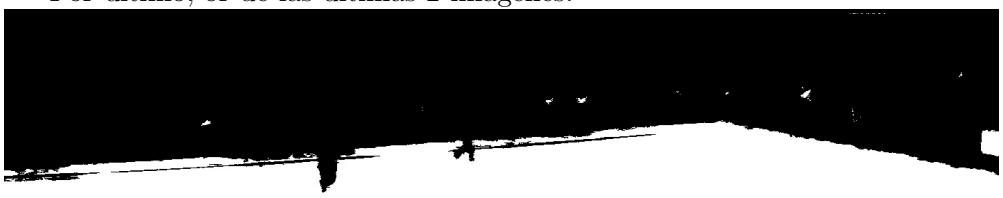
Al analizar el funcionamiento del código anterior se tiene: 1) lugar se realiza un llenado de color blanco en el punto (0,0). Es muy importante seleccionar el punto de inicio cuidadosamente. Siempre se debe seleccionar en el punto del background (negro), que al ser un analizador de videos de futbol, se sabe que en el punto (0,0) está la gradería que siempre constituye a nuestro background negro. Al rellenar el background negro, de color blanco, básicamente se obtiene una imagen blanca, con los blobs negros a rellenar 2) Se hace un not de la imagen original, obteniendo la imagen binaria invertida. 3) Finalmente se hace un or de imágenes entre las imágenes obtenidas del punto 1) y punto 2), y se obtiene la imagen rellenada". Las 3 imágenes mencionadas se observan en las siguientes 3 imágenes:

Primero la imagen con el fondo rellenado:



Luego la imagen original invertida:

Por último, or de las últimas 2 imágenes:



Particularmente en este método no hubo ningún problema a la hora de la implementación debido a la mucha documentación que había, y a una comunidad activa y grande de opencv.

3.3. bwareopen() - opencv

Este método tiene como objetivo eliminar las regiones espurias. Estas pequeñas regiones corresponden a pequeños blobs que estarán en la gradería del campo de futbol, debido a que entraron dentro del umbral considerado "verdoso". Tiene como entrada una imagen binaria (que previamente fue rellenada con imfill()) y su salida es una imagen con las regiones espurias rellenadas.

Básicamente el algoritmo tiene 3 pasos importantes: 1) encontrar todos los contornos en la imagen. Contornos son sinónimos de blobs. Se hace de la siguiente manera:

```
Imgproc.findContours(clonedImage, contours, hierarchy,
    Imgproc.RETR_CCOMP,Imgproc.CHAIN_APPROX_SIMPLE, new Point(0,0));
```

.

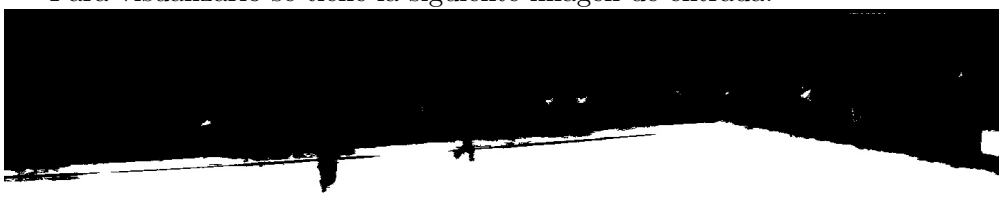
2) luego de tener todos los contornos, se recorre la lista de contornos, y si es el área del contorno es menor a una constante, este se considera una región espuria a eliminar. Hay que eliminar cuidadosamente el tamaño que se considera un blob como región espuria". Se hace de la siguiente manera:

```
if (Imgproc.contourArea((contours.get(i))) < Constants.MAXSIZE){

}
```

3) Una vez encontradas todas las regiones espurias, se rellenan con un color negro. Se hace con el siguiente código:

```
Imgproc.drawContours(polishedImage, littleContours, -1, Constants.BLACK,-1);
```

Para visualizarlo se tiene la siguiente imagen de entrada:



Luego de aplicar bwareopen() se tendría la imagen sin regiones espurias:

Particularmente en este método no hubo ningún problema a la hora de la implementación debido a la mucha documentación que había, y a una comunidad activa y grande de opencv.

3.4. detectField() - opencv

Este algoritmo recibe una imagen de un partido de futbol en formato rgb, y devuelve una imagen binaria con el campo de futbol reconocido.

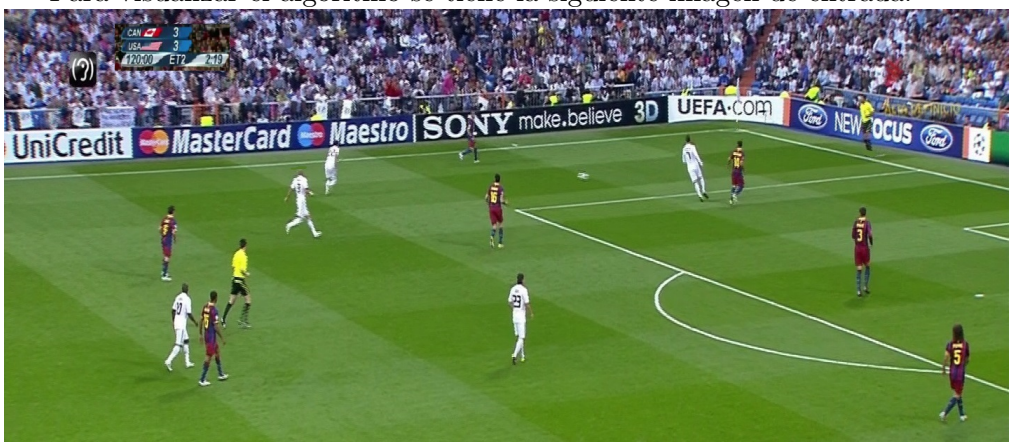
```
public static Mat detectField(Mat pImage){
  //detects the soccer field
   Mat rgb = pImage.clone();
   Mat hsv = rgb2hsv(rgb);
   Mat greenMask = greenMask(hsv);
    Mat dilatedImage = dilate(greenMask);
    Mat filledImage = imfill(dilatedImage);
    Mat polishedImage = bwareaopen(filledImage);
```

```
        Mat finalImage = finalTouch(polishedImage);
        Mat imageWithoutScore = removeScore(finalImage);
        return imageWithoutScore;
    }
```

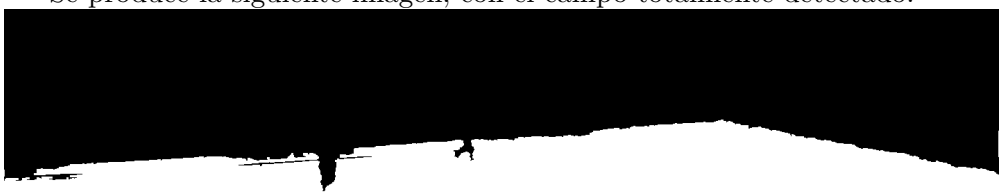
.

El algoritmo tiene los siguientes pasos: 1) convertir la imagen de entrada rgb en una imagen hsv, 2) obtener una máscara binaria de pixeles "verdososçon greenMask(), 3) rellenar hoyos por posibles jugadores con imfill(), 4) se pule la imagen para eliminar ruido con dilate(), 5) se eliminan regiones espurias por posibles zonas "verdosas.[en] la gradería con bwareopen(), 6) es posible que queden hoyos no rellenados desde la máscara binaria, por lo que se saca el complemento de la imagen y de nuevo se eliminan regiones espurias con bwareopen(), a esto método le llamamos finalTouch(), y 7) por último se elimina el marcador del partido (si era verde), con removeScore() (básicamente se dibuja un rectángulo negro que esta alambrado a la esquina superior izquierda, lo cual es una suposición y limitación que se debe tomar en cuenta).

Para visualizar el algoritmo se tiene la siguiente imagen de entrada:



Se produce la siguiente imagen, con el campo totalmente detectado:



Particularmente en este método no hubo ningún problema a la hora de la implementación debido a la mucha documentación que había, y a una comunidad activa y grande de opencv. Que no se haya presentado ningún problema no quiere decir que no requirió mucho tiempo de aprendizaje por la poca experiencia que se tenía en el tema.

4. Detalles de problemas encontrados con las herramientas

4.1. Problemas en desarrollo

La curva de aprendizaje de java (Enfocado en desarrollo web) ha sido el mayor reto en nuestro desarrollo. Se dedico todo un día de investigación para tener un mayor entendimiento sobre el tema, sin embargo si se debe notar que la mayoría de la información encontrada no era muy reciente. Tras tener un alto conocimiento del tema, se busco la manera de implementar el server en Java integrando

a Eclipse EE, para facilitar las pruebas durante el desarrollo. Sin embargo, al realizar la primera conexión, ya simplifica bastante la tarea de la implementación de las otras funcionalidades. Para la parte lógica y del usó de OpenCV, no fue tan complicado debido al la cantidad de documentación al respecto.

4.2. Problemas en organización del equipo de trabajo

La organización del equipo de trabajo está pasando por una etapa de descubrimiento de herramientas, por lo cual es necesario ampliar el tiempo de planeación y coordinación. Por otra parte, la herramienta seleccionada para gestión del proyecto genera exceso de correos.

4.3. Problemas en concepto del software

El equipo de trabajo no posee experiencia en el área de la detección de objetos en imágenes por lo cual ha sido necesario pasar por un proceso de aprendizaje fuerte. A raíz de esto, ha sido necesario realizar consultas con expertos en el tema, requiriendo así un mayor tiempo en la etapa de implementación y diseño del proyecto.

5. Github

El código del proyecto, se encuentra en el siguiente link: `https://github.com/joshuamataaraya/SoccerAnalysis`

6. Gestion del proyecto

Para gestionar el proyecto se está usando una herramienta en línea llamada Zoho. Dentro del git se encuentra un archivo de Excel que contiene la bitácora de lo que se ha realizado en la iteración 1 del proyecto.

Se puede observar que existen unas tareas relacionadas con la implementación denominada "Posible Players Location". Estas tareas no fueron completadas, sin embargo eran de prioridad baja en esta iteración, por lo cual estas tareas van a ser movidas a la siguiente iteración. La prioridad baja de estas tareas se debe a que se pretendía abarcar más de lo establecido como meta en esta iteración.