UNIVERSIDAD DE COSTA RICA
SISTEMA DE ESTUDIOS DE POSGRADO

RASTREO DE JUGADORES DE FÚTBOL MEDIANTE GRAFOS MULTIPARTITOS UTILIZANDO VIDEOS DE ULTRA ALTA DEFINICIÓN.

Tesis sometida a la consideración de la Comisión del Programa de Posgrado en Ingeniería Eléctrica para optar al grado y título de Maestría Académica en Ingeniería Eléctrica

MARCO VILLALTA FALLAS

Ciudad Universitaria Rodrigo Facio, Costa Rica

2019

Dedicatoria

A mi familia,

Gracias

Agradecimientos

Deseo expresar mi gratitud a todos aquellos que colaboraron en el desarrollo de este trabajo. A mi profesor tutor, Dr. rer. nat. Francisco Siles Canales por su apoyo y guía. A Fabian Mora, Marcos Jimenez, Bryan Cervantes y a los miembros de ACE/PRIS-Lab por la colaboración en las etapas de adquisición y procesamiento de datos. También agradezco a mis compañeros de la maestría por su acompañamiento durante esta etapa.

Esta tesis fue aceptada por la Comisión del Programa de Estudios de Posgrado en Ingeniería Eléctrica de la Universidad de Costa Rica, como requisito parcial para optar al grado y título de Maestría Académica en Ingeniería Eléctrica.

Dr. Marvin Coto Jiménez
Representante del Decano
Sistema de Estudios de Posgrado

Dr. rer. nat. Francisco Siles Canales
Director de Tesis

Dr. Ing. Juan Luis Crespo Mariño
Asesor

Dr. Álvaro De La Ossa Osegueda
Asesor

Dr. rer. nat. Federico Ruiz Ugalde
Representante del Director
Programa de Posgrado en Ingeniería Eléctrica

Marco Villalta Fallas
Candidato

Tabla de contenido

Resumen

Este trabajo describe un algoritmo de rastreo para jugadores de fútbol basado en gráficos multipartitos diseñados para el procesamiento de un gran volumen de datos. El algoritmo propuesto utiliza varias características como: contornos, información cromática y dinámica, para la asociación de datos dentro de un gráfico multipartito para resolver oclusiones y rastrear a jugadores de fútbol. La implementación paralela del algoritmo realiza un esquema consumidor-productor para superponer el tiempo de procesamiento de los dos procedimientos principales del algoritmo: segmentación y rastreo; así como un patrón de comunicación de envío y recepción para propagar las identidades de objetos. Mostramos cómo un sistema híbrido de paralelización de datos y tareas mejora el tiempo de ejecución para videos 4K, logrando una aceleración igual a 19.24 y una velocidad de procesamiento de 21.71 FPS con 128 subprocesos. Utilizando la base de datos ISSIA se obtuvieron valores similares de las métricas de FP y FN con una velocidad de rastreo superior.

Abstract

This work describes a tracking algorithm for football players based on multipartite graphs designed for the processing of high volume of data. The proposed algorithm use several characteristics such as: contours, chromatic and dinamic information, for the association of data within a multipartite graph to solve oclusions and track football player. The parallel implementation of the algorithm performce a consumer-producer scheme to overlap the computing time of the two main procedures of the tracking algorithm: segmentation and tracking; as well a send-and-receive communication pattern to propagate the blob identities. We show how an hybrid system of data and task parallelization improves the execution time for 4K videos, achieving a speedup equal to 19.24 and a processing speed of 21.71 FPS with 128 threads. Using the ISSIA database, similar values were obtained from the FP and FN metrics with a higher tracking rate.

Lista de tablas

Lista de figuras

Lista de algoritmos

Acrónimos

API	Application Programming Interface (Interfaz de programación de aplicaciones)
CPU	Central Processing Unit (Unidad de Procesamiento central)
CUDA	Computer Unified Device Architecture (Arquitectura Computacional Unificada de Dispositivos)
DMA	Dynamic Memory Access (Acceso Dinámico de Memoria)
FHD	Full High Definition (Alta Definición Completa)
FPS	Frames Per Second (Cuadros Por Segundo)
GPU	Graphical Processing Unit (Unidad de Procesamiento Gráfico)
HD	High Definition (Alta Definición)
HSV	Hue Saturation Value (Tono Saturación Valor)
LP	Programming Language (Lenguaje de Programación)
MP	Multi-Processing
MPI	Message Passing Interface
MPGT	Multipartite Graph Tracker
PX	Pixel (Pixel)
RAM	Random Access Memory (Memoria de Acceso Aleatorio)
RGB	Red Green Blue (Rojo Verde Azul)
SD	Standard Definition (Definición Estándar)
SMP	Symmetric Multi-Processing (Multiprocesamiento Simétrico)
TV	Television (Televisión)

UHD Ultra High Definition (Ultra Alta Definición)

Capítulo 1

Introducción

El fútbol es uno de los deportes colectivos más populares en el mundo, con un aproximado de 3.5 billones de aficionados y de los deportes más complejos actualmente practicados (Wood, 2017). Tal popularidad ha llevado a que en los últimos años, existan investigaciones científicas con un creciente interés por diseñar e implementar, sistemas computacionales que favorezcan el análisis de resultados. Una amplia gama de aplicaciones, pueden estar soportadas por los sistemas computacionales mencionados anteriormente, por ejemplo: el resumen de juegos mediante movimientos destacados, los análisis estratégicos, tácticos y estadísticos de equipos o jugadores individuales; el desarrollo de mejores estrategias de entrenamiento y la verificación de las decisiones arbitrales.

Las aplicaciones que miden las métricas de desempeño de jugadores o equipos, tienen el potencial de revelar información que no es obvia a simple vista. Estos sistemas pueden medir distancia recorrida, velocidad de movimientos o posición del balón. Esta información es cada vez más popular en la ciencia del deporte, al ser utilizada luego para realizar una evaluación física, detección de fatiga, análisis de los oponentes y evaluación del desempeño táctico (Radakovic, Dopsaj & Vulovic, 2015).

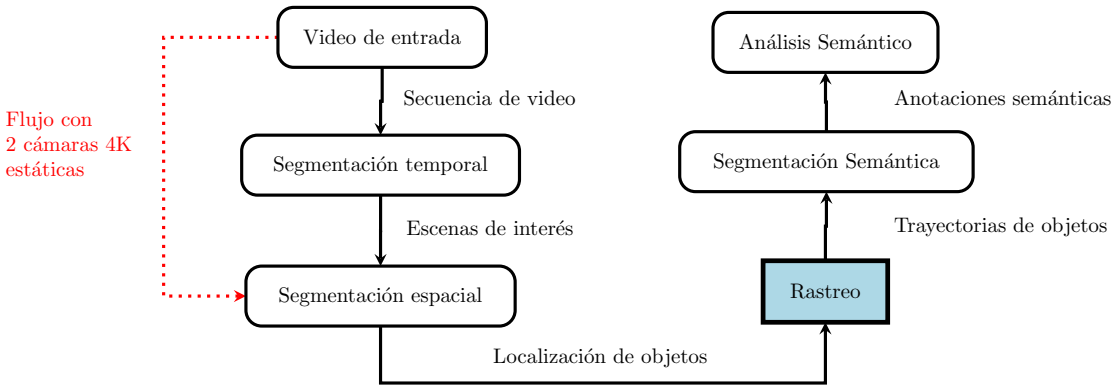


Figura 1.1: Arquitectura de ACE cuyos cuatro primeros bloques corresponden a la etapa de percepción y los últimos dos bloques corresponden a la etapa cognitiva.

(a) Acercamiento 1 **(b)** Público

(c) Transición de repetición **(d)** Toma a lo ancho

(e) Toma lejana **(f)** Tiro de esquina

Figura 1.2: Diferentes tomas de una transmisión de TV. Para un sistema de análisis deportivo solo las tomas lejanas aportan información sobre la dinámica del juego.

Para lograr lo anterior, el rastreo de jugadores requiere detectar múltiples jugadores en un video, encontrando sus posiciones en intervalos regulares y asociando la información espacial y temporal para extraer las trayectorias. Esta tarea no es trivial por los patrones de movimientos impredecibles, además de que los jugadores se ven muy parecidos y frecuentemente se encuentran en la lucha por la posesión del balón. Se debe considerar dentro de los retos de este tipo de rastreo, los factores externos como las condiciones ambientales: lluvia, cambios de iluminación y sombras.

En el Laboratorio de investigación en reconocimiento de patrones y sistemas inteligentes (PRIS-Lab), se desarrolla un sistema para cumplir las necesidades mencionadas anteriormente. Dicha plata-

Figura 1.3: Detección y localización de jugadores resultante de la segmentación espacial de ACE.

forma se llama ACE, es una plataforma computacional de análisis de videos digitales de fútbol, que genera modelos abstractos de un partido de fútbol, dotando de un análisis automatizado de un encuentro de fútbol a científicos del movimiento humano, entrenadores, equipos y público en general. Esta plataforma se implementa en una arquitectura multicapa, como se observa en la figura 1.1 y consta de dos etapas principales: una de percepción y la otra cognitiva.(F. Siles & Saborío, 2015).

La primera etapa de la plataforma ACE, está compuesta por tres bloques: la segmentación temporal, donde se hace la detección y clasificación de escenas, produciendo escenas candidatas, que contienen jugadores que se puedan rastrear en una toma lejana.

En la figura 1.2 se muestra varias tomas que se encuentran típicamente en un video de una transmisión de televisión, para el efecto de un sistema de análisis deportivo las tomas 1.2a, 1.2f, 1.2c, 1.2d y 1.2b no generan información útil y por lo tanto son filtradas por la segmentación temporal, dejando solo las tomas lejanas como la figura 1.2e.

Otro bloque es la segmentación espacial que se encarga de la detección y localización de objetos en cada uno de los cuadros de las escenas, como se muestra en la figura 1.3, donde el rastreo es responsable de generar las trayectorias de los objetos.

La segunda etapa, está formada por dos bloques, recibe como entrada las trayectorias de los jugadores, árbitros y el balón, producto de la etapa de percepción. Se encarga de realizar una reinterpretación de estas trayectorias, para generar un modelo abstracto de mayor nivel. En el bloque de la segmentación semántica se hace la detección de estas acciones y la clasificación de eventos. El análisis semántico es donde se generan las estadísticas, el mapa de ocupación y demás información; es la salida del sistema (Francisco Siles, 2014).

La investigación de este trabajo se centra en la primer etapa de la plataforma ACE, específicamente en el bloque que realiza la detección y rastreo de jugadores. A diferencia de lo que se ha ido realizando en la plataforma, que ha sido usar videos provenientes de transmisiones de televisión con una calidad inferior a "HD", se utilizará como fuente de datos la concatenación de los videos generados por dos cámaras estáticas de ultra alta definición.

La robustez y precisión del rastreo está en función de las etapas de segmentación espacial y temporal, que extraen las características que se usan en el rastreo. En la figura 1.1 se muestra el flujo alterno con la configuración de dos cámaras estáticas. No utiliza el bloque de segmentación temporal, debido a que se no existen escenas sin información relevante, este flujo y las cualidades de las imágenes pretenden reducir fuentes de imprecisión. Al tener dos imágenes con una mayor resolución, mejora la calidad de las características y permite tener la descripción de todo el sistema.

Otra diferencia relevante de esta trabajo con investigaciones previas en el PRIS-Lab, es el uso de grafos multipartitos. Para comprender adecuadamente la idea anterior es importante definir conceptos básicos de grafos. Un grafo es una representación abstracta consistente en un conjunto de nodos, los cuales representan objetos y un conjunto de bordes, los cuales representan asociaciones entre pares de objetos (Phillips, 2015). Un grafo se considera multipartito si el conjunto de nodos se pueden dividir en subconjuntos no vacios, de tal forma que los nodos en un mismo subconjunto no estén conectados (Sitton, 1996).

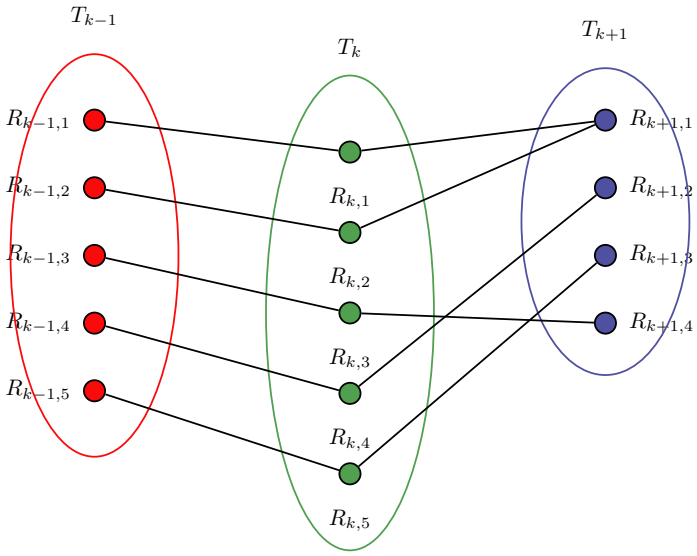


Figura 1.4: Grafo multipartito minimizado compuesto por tres grafos

En la figura 1.4 se muestra un grafo multipartito compuesto por tres subconjuntos, donde el primer y segundo subconjunto tienen cinco nodos, mientras el subconjunto restante tiene cuatro nodos.

En las investigaciones previas del PRIS-Lab, el rastreo de objetos se hace cuadro a cuadro por medio de una asociación con grafos bipartitos. En un grafo bipartito, un subconjunto representa el estado del sistema en el cuadro anterior y el otro subconjunto el estado del sistema en el cuadro presente. Cada jugador esta asignado a un nodo, donde su posición actual se asocia a la posición anterior por medio de un borde que encapsula varias propiedades descriptivas del jugador. Al utilizar grafos multipartitos en este trabajo, se pretende tener una mejor representación del sistema, al encontrar un óptimo que lo describa de forma más apropiada y que mejora el rastreo de los jugadores.

1.1. Problema

Rastrear de manera precisa a los jugadores de fútbol con una configuración de dos cámaras 4k estáticas, a partir de información temporal de múltiples cuadros consecutivos, representados en grafos multipartitos.

1.2. Hipótesis

A través de un algoritmo de rastreo, utilizando grafos multipartitos cuyos nodos corresponden a jugadores y los bordes, a vectores que describen las diferencias entre caracerísticas de cada jugador: la función de distancia Hausdorff entre contornos, la distancia bhattacharyya de los histogramas en la representación HSV y la posición, a partir de un modelo cinemático de velocidad constante; se obtendrá al menos un 75 % de la métrica F1 Score y una velocidad de procesamiento superior a 14 FPS.

1.3. Propuesta

1.4. Objetivos

Objetivo General

Crear un algoritmo automatizado de rastreo y clasificación de jugadores, utilizando grafos multipartitos y videos de ultra alta definición.

Objetivos Específicos

1. Desarrollo de una plataforma de rastreo automatizado de jugadores.
2. Diseño e implementación del algoritmo de rastreo y clasificación de jugadores.
3. Validación del algoritmo mediante comparación con datos de referencia anotados manualmente.

Capítulo 2

Antecedentes

La investigación sobre el seguimiento de objetos está bien arraigada y se aplica a una amplia gama de dominios (Yilmaz, Javed & Shah, 2006). En términos generales el rastreo de objetos consiste en la estimación de trayectorias de objetos en movimiento en una secuencia de imágenes (Goszczynska, 2011).

2.1. Soluciones Comerciales

En el transcurso de la última década, las soluciones comerciales para el análisis deportivo, han enfocado sus productos a distintos clientes con necesidades particulares. Esto refleja la forma en que sus sistemas son utilizados. Dentro de estas soluciones de análisis deportivos, cabe mencionar aquellas enfocadas a deportes y a campos específicos. También hay soluciones que proveen un conjunto de herramientas, que se pueden utilizar para distintos deportes.

En el pasado han surgido y desaparecido empresas, dedicadas a proveer herramientas para el estudio de los deportes. A continuación, se mencionan las que se encuentran activas y tienen relevancia en la investigación de este trabajo.

Kizanaro

Kizanaro es una compañía de origen Uruguayo, creada en el 2006 que se dedica a brindar servicios de tecnología aplicada al fútbol. Este software es utilizado por las selecciones de Uruguay, Paraguay, Venezuela y Colombia, además de varios equipos profesionales de las ligas de los países anteriores. Se usa para generar análisis y estadísticas de cada partido. La compañía indica que su software clasifica y desglosa todas las acciones del partido, permitiendo visualizarlas en el video. Entre 24 y 48 horas después de enviar el video a la compañía, el director técnico recibe un informe con detalles del partido sobre el equipo analizado y un archivo para visualizar el análisis en base al video (Technology, 2017).

Figura 2.1: Anotación de eventos con Kizanaro

(a) Cámaras

(b) Rastreo supervisado

Figura 2.2: Sistema de rastreo computacional Tracab para fútbol.

Por la información que indica el fabricante del software, es necesario hacer la anotación manual de los eventos como se observa en la figura 2.1 y no realiza un rastreo automatizado de los jugadores.

Tracab

Tracab inició en 2003 en Suecia. Se consideran el sistema más avanzado basado en cámaras para el rastreo de jugadores y balón en el mercado comercial, se encuentra instalado en aproximadamente 300 estadios, siendo la tecnología de rastreo oficial de las ligas de fútbol inglesa, alemana y española. Además, se utiliza en competiciones de la UEFA. Este sistema no se limita solo al fútbol, también funciona para baseball, cricket, tenis y fútbol americano.

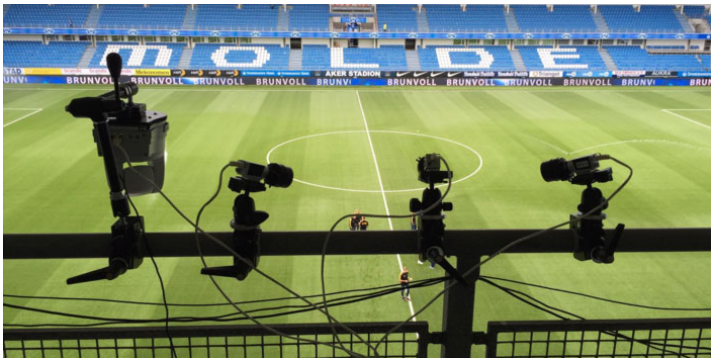


Figura 2.3: Configuración del sistema multi-cámara SportVU de STATS

En la figura 2.2 se muestra parte de los componentes del sistema. La arquitectura del sistema de cámaras consiste en dos unidades múltiples (8 cámaras HD individuales) posicionadas en las partes superiores de los estadios, un servidor y una computadora portátil. Realiza un rastreo tridimensional aprovechando la configuración estereoscópica de las unidades de cámaras con una frecuencia de muestreo de 25 cuadros por segundo. El rastreo es supervisado por un operador en el transcurso del partido como se observa en la figura 2.2b(Corporation, 2017).

STATS

STATS es una compañía que tiene más de 35 años de trayectoria, dedicada al campo del deporte. Ofrece soluciones a los distintos actores: aficionados, empresas de transmisión de eventos y equipos, por lo tanto tiene varios productos dependiendo de las necesidades y el deporte.

Para el fútbol tiene las siguientes categorías: monitoreo de atletas, análisis de videos, análisis de juegos y captura de datos. Con respecto a la captura de datos el componente que disponen se llama SportVU. Indican que el sistema SportVU de rastreo ofrece estadísticas de rendimiento mediante la extracción y el procesamiento de coordenadas de los jugadores y la pelota a través de cámaras “HD”, así como software sofisticado y algoritmos estadísticos(LLC., 2017).

La instalación de este sistema tiene dos arquitecturas de cámaras. Una configuración de las cámaras tiene cuatro cámaras estáticas de alta definición que se muestra en la figura 2.3. La otra configuración agrega otras tres cámaras estáticas para tener información del posicionamiento tridimensional.

(a) Cámaras

(b) Representación panorámica.

Figura 2.4: Sistema de cámaras para la captura panorámica del campo de la plataforma Match Analysis

MatchAnalysis

Match Analysis se creó en el 2000 en los EUA, con instalaciones de recolección de datos en EUA y México. Provee un conjunto de herramientas para el análisis de videos y un servicio de librería digital, para proveer información de desempeño y registro de datos, que muestren patrones tácticos de los equipos contrarios.

Las herramientas de Match Analysis, son usadas en las ligas profesionales de EUA y México. En la información oficial indican que recogen los detalles de los partidos en tiempo real y sincronizan los datos con el video. En la figura 2.4 se muestra el sistema de cámaras estáticas que generan imágenes de 8k como en la subfigura 2.4b (Inc., 2017). De forma implícita dan a entender en la información disponible al público que no realizarán rastreo automatizado. Una diferencia importante con Tracab en la arquitectura de cámaras es que también ofrecen un sistema portátil.

Debido a la naturaleza comercial y a la propiedad intelectual de los algoritmos utilizados por estas compañías, no se puede tener acceso a la documentación que describe el funcionamiento de estos sistemas y no existen publicaciones científicas que los describan.

2.2. Literatura Profesional

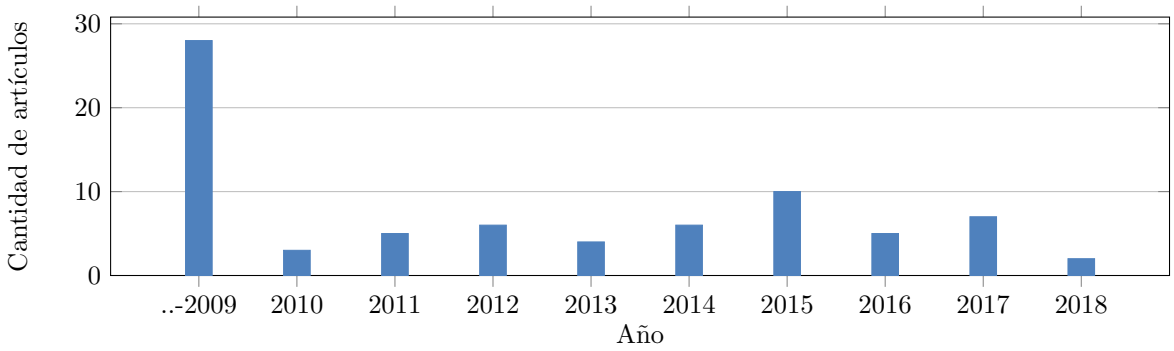


Figura 2.5: Cantidad de artículos agrupados por año sobre rastreo de fútbol

A continuación se presenta una revisión de la literatura profesional, de los estudios relevantes al dominio del rastreo de jugadores de fútbol y se muestran organizados por año como un histograma en la figura 2.5, todas aquellas publicaciones previo del 2009 se agrupan en un solo conjunto por simplicidad. El problema del rastreo de múltiples jugadores en un deporte ha sido abordado con diferentes enfoques que se pueden agrupar como: Métodos determinísticos, métodos estadísticos y asociación de datos (Yilmaz et al. 2006).

Métodos determinísticos

Los métodos determinísticos, son aquellos que utilizan características visuales de una forma determinista, para rastrear a los jugadores.

Las plantillas de color, son usados en propuestas como en (Seo, Choi, Kim & Hong, 1997) y (Bebie & Bieri, 1998), donde se utilizan estas plantillas para comparar, las distintas regiones en un cuadro y en caso de coincidir, se determina como la detección del objeto. (Lefèvre, Fluck, Maillard & Vincent, 2000) utiliza la técnica de contornos activos en dos partes: la inicialización y un deformador *snake*, para aprovechar la características del contorno. La idea de estimar la densidad de funciones como el *mean-shift* (Comaniciu & Meer, 2002) es utilizada en (Hu, Chang, Wu & Chi, 2011) donde se usan las características de color en histogramas aprovechando la ventaja de este descriptor para rastrear objetos no rígidos. Para mejorar el desempeño del rastreador se puede incorporar información de textura y localización como en (Misu, Naemura, Zheng, Izumi & Fukui, 2002) y (Pers & Kovacic, 2001). En (Hare,

Saffari & Torr, 2011) se utiliza un *support vector machine* (SVM) para que el algoritmo aprenda la apariencia del objeto y de esta forma se pueda adaptar a los cambios productos de condiciones ambientales.

El problema de estos métodos consiste en una pérdida del rastro cuando se dan la interacción de objetos con similares características, por lo tanto estos métodos suelen perder el rastro cuando los jugadores de un mismo equipo se obstruyen o incluso si se encuentran cerca. Los algoritmos de los métodos determinísticos son usados frecuentemente por algoritmos más complejos de otros tipos.

Métodos probabilísticos

Los métodos probabilísticos, son aquellos que utilizan el marco de referencia bayesiano y sus estimaciones para realizar el rastreo de múltiples jugadores. Los movimientos aleatorios pueden ser rastreados secuencialmente por medio de estimaciones de Monte Carlo, conocidos también como filtros de partículas. Los métodos que utilizan filtros de partículas muestran una tendencia a ser lo más utilizados en la actualidad para el rastreo de jugadores.

La técnica de filtro de partículas se utiliza en (Czyz, Ristic & Macq, 2005) como una forma de encapsular en un solo estado a todos los jugadores, sin embargo es una forma inadecuada, debido a que el rastreo erróneo de un jugador afecta la estimación total. Una forma de solucionar este problema es el uso de rastreadores de filtros de partículas individuales como en (Dearden, Demiris & Grau, 2006), (Ok, Seo & Hong, 2002), (Kristan, Perš, Perše & Kovačič, 2009), (Morais, Goldenstein, Ferreira & Rocha, 2012), (Morais et al. 2014), (Petsas & Kaimakis, 2016) y (Hess & Fern, 2009).

Uno de los problemas con los métodos basados en filtros de partículas, es que el rastreo de jugadores de fútbol se va degradando con el tiempo, debido a las constantes oclusiones y acercamientos entre jugadores que propician que las partículas se asignen a jugadores incorrectos. Esto sucede, debido a que las particulas tienden a poblarse cerca de picos con distribuciones posteriori. En la práctica, las particulas de un objeto cambian de lugar fácilmente a una región adyacente cuya región tiene mayor verosimilitud (Seo & Hong, 2004). Con el motivo de reducir esta degradación, (Itoh, Takiguchi & Ariki, 2012) utiliza un filtro de particulas guiado por un grafo espacio-temporal para hacer más robusto el rastreo de jugadores a oclusiones.

Asociación de datos

Los métodos de asociación de datos son aquellos que, una vez detectados los objetos, plantean el problema de rastreo a partir de una asociación de datos que busca una solución óptima a esta relación.

Los primeros enfoques de asociación de datos usualmente confiaban en filtros Kalman. Debido a su naturaleza recursiva, cuando se utilizan para rastrear objetos en escenas llenas de personas o con oclusiones, son propensos a cambio de identidades, un efecto difícil de recuperar (Shitrit, Berclaz & Fleuret, 2014).En la actualidad se utiliza esta técnica en (Petsas & Kaimakis, 2016) y (Sahbani & Adiprawita, 2016), otras técnicas se abordan a continuación.

En (Gedikli, Bandouch, Hoyningen-Huene, Kirchlechner & Beetz, 2007) se usa un rastreador de múltiples hipótesis, para crear asociaciones entre las posiciones observadas y previas de un jugador. (Xu, Orwell & Jones, 2004) utiliza un filtro de datos de probabilidades conjuntas, para relacionar a un jugador en cuadros consecutivos. (Iwase & Saito, 2004) realiza el rastreo utilizando homografía que representa la relación geométrica entre varias cámaras. (Shitrit, Berclaz & Fleuret, 2011) crea un mapa de ocupación probabilístico para cada uno de los jugadores como grafos acíclicos, encontrando la solución óptima con programación lineal.

En (Soomro, Khokhar & Shah, 2015) se contruye un grafo optimizado por equipo, con los datos de la formación de los equipos, para asociar trayectorias entre secuencias temporalmente separadas. En (Figueroa, Leite, Barros, Cohen & Medioni, 2004) y (Figueroa, Leite & Barros, 2006) se realiza un rastreo de jugadores, con múltiples cámaras utilizando representaciones gráficas, donde los nodos corresponden a *blobs* obtenidos de la segmentación de la imagen y los bordes representan las distancias entre ellos. En (WL, JA, JJ & KP, 2013) se realizan asociaciones jerárquicas de datos utilizando las condiciones de un modelo de movimiento y grafos bipartitos para rastrear a los jugadores. Esta idea es considerada en (Francisco Siles, 2014) combinando, en los bordes las características de color, textura, forma y la información de un modelo cinemático de los jugadores de fútbol. En (Manafifard, Ebadi & Moghaddam, 2015) se utiliza un grafo de nodos cercanos, el tamaño de jugadores adyacentes, las características cromáticas y las dimensiones del nodo para predicir y separar las oclusiones encontrando las trayectorias óptimas usando el algoritmo de optimización de enjambre de partículas.

En la literatura profesional, se considera que estos métodos agregan una complejidad computacional adicional, al tener que realizar el proceso de asociación y descarte de relaciones, además de la detección, considerándolos imprácticos. Sin embargo, las tendencias de las tecnologías de computación de alto desempeño, abren la posibilidad de considerar este tipo de algoritmos como nuevos métodos factibles.

Desempeño y velocidad

Ciertamente, el objetivo principal de un rastreador de jugadores de fútbol es la identificación precisa y el seguimiento de cada objeto en el campo, sin embargo, el costo computacional y la latencia son aspectos de gran relevancia, por lo tanto se consideran en la literatura profesional.

Estas consideraciones se pueden encontrar desde el año 2000 donde Lefèvre et al. (Lefèvre et al. 2000) crea un método basado en *fast snake* donde obtiene un tiempo de procesamiento de 2 segundos por cuadro, con una implementación en Matlab para una secuencia de 100 imágenes de 24 bits, con ancho y alto de 384 y 288 píxeles respectivamente. Otros trabajos muestran un rendimiento de velocidad similar al utilizar Matlab como plataforma de desarrollo, (de Vos & Brink, 2009) mide 4 FPS de la etapa de detección de movimiento para un rastreador de filtro de partículas, (Yuan, Emmanuel & Fang, 2019) obtiene 3 FPS para su rastreador de objetos visual, (Martín & Martínez, 2014) 3.3 FPS en un sistema semi-supervisado y (Najafzadeh, Fotouhi & Kasaci, 2015) para un rastreador que se basa en un filtro de Kalman adaptativo. Ma et al. (Ma, Feng & Wang, 2018) usa redes siamesas completamente convolucionales para la extracción de características y el solucionador de oclusiones; este algoritmo se implementa en Matlab con la caja de herramientas MatConvNet y funciona a 4 FPS con un procesador Intel Core i7-4720HQ y una tarjeta gráfica NVIDIA GeForce GTX 960M GPU. Matlab es un lenguaje de programación apropiado para desarrollar y crear prototipos de algoritmos, pero muestra valores más bajos de cuadros por segundo en las implementaciones de los algoritmos de rastreo, esto se debe a que es un lenguaje interpretado que utiliza un compilador JIT para traducir un *script* a código de máquina.

Sentioscope utiliza un enfoque para un sistema de rastreo de jugadores que se basa en un filtro de partículas de campo (Baysal & Duygulu, 2016). Este algoritmo se implementa en C++ con una GPU mediante una técnica de paralelización de tareas, ejecutandolas en diferentes subprocesos: adquisición de imágenes, exposición automática, ajuste de luz, extracción de primer plano y cálculo HOG; mientras

que las tareas de clasificación y seguimiento del jugador se ejecutan en paralelo con un patrón de unión. El sistema se ejecuta en una computadora portátil con un CPU Intel i7 de 4 núcleos que alcanza una razón de 14 FPS.

Existen algoritmos de asociación de datos similares a los algoritmos que se fundamentan en la teoría de grafos, por ejemplo (Shitrit et al. 2014) utiliza una formulación de flujo de red como un problema de optimización de programación lineal, este algoritmo se implementa en C++ usando STL. Utilizan una PC de 3 GHz y un único núcleo, cuyo resultado es una velocidad de 3,95 FPS para la versión MCNF del algoritmo con la base de datos ISSIA.

La etapa de procesamiento de imágenes tiene un gran impacto en la velocidad de ejecución de los rastreadores. Por lo tanto, la paralelización de la etapa de segmentación espacial, donde se extraen los jugadores de fútbol, se discute en (F. Siles & Saborío, 2015) para la plataforma computacional ACES, donde plantean un método de paralelización basado en múltiples subprocesos de OpenMP. Realizan varias pruebas con videos de definición estándar y obtienen un factor de aceleración de 4 con una eficiencia de 0.1 al ejecutarse con 8 subprocesos. Este algoritmo se implementa en C ++ con la biblioteca OpenCV.

Es interesante como varios artículos científicos mencionan que los tiempos de ejecución pueden mejorarse utilizando técnicas de paralelización, sin embargo, no consideran utilizar los sistemas distribuidos y GPGPU (Najafzadeh et al. 2015), (Hoseinnezhad, Vo, Vo & Suter, 2012), (Morimitsu, Cesar & Block, 2015). Todos los trabajos relacionados se ejecutan en computadoras personales y los valores de FPS presentados por estos muestran que su implementación y recursos no son suficientes para lograr un tiempo de procesamiento ideal (igual o superior a 30 FPS).

Capítulo 3

Marco teórico

3.1. Reconocimiento de patrones y procesamiento de imágenes

Reconocimiento de patrones consiste en tomar una acción o decisión basado en el tipo de patrón observado en los datos de entrada. Los patrones reconocidos en los datos se obtienen a partir de los procesos de segmentación, extracción de características y descripción donde cada objeto queda representado por una colección de descriptores; estas fases se muestra en la figura 3.1. En la gran mayoría de aplicaciones, las variables de entrada son pre-procesadas ó segmentadas para transformarlas a un nuevo espacio de variables donde, el problema de reconocimento de patrones es más factible de resolver (Bishop, 2006).

La etapa inicial, consiste en extraer y seleccionar las características (atributos) necesarias para construir un vector n-dimensional que permita realizar ya sea clasificación, agrupamientos, encontrar reglas de asociación ó modelar series temporales; este procesamiento de información se conoce como extracción de características.

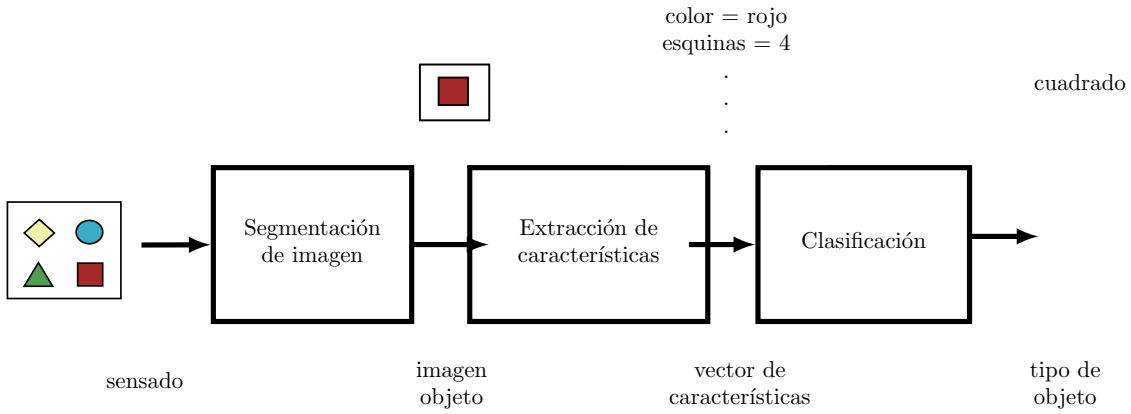


Figura 3.1: Las tres fases de un sistema de reconocimiento de imágenes y su resultado intermedio y final.

En la clasificación se trata de asignar las diferentes partes del vector de características a categorías finitas y discretas. Debido a la imposibilidad de tener un clasificador perfecto, se suele utilizar una forma más práctica, la determinación de una probabilidad para cada una de las categorias. La abstracción que ofrece el vector de características sobre los datos de entrada habilita el desarrollar teorias de clasificación con un dominio más amplio (Duda & Stork, 2001).

El agrupamiento, que son un conjunto de algoritmos de clasificación no supervisada, se encarga de organizar una colección de objetos en clases o grupos, de forma tal que los objetos pertenecientes a un mismo grupo sean lo suficientemente similares como para poder inferir que son del mismo tipo y los objetos pertenecientes a grupos distintos sean lo suficientemente diferentes como para poder afirmar que son de tipos diferentes (Gandón, Pons & Ruiz-Shulcloper, 2012).

Muchas de las diferentes técnicas usadas para resolver los problemas de reconocimiento de patrones se pueden agrupar en dos enfoques generales: el enfoque discriminante y el sintáctico. En el enfoque discriminante un conjunto de características medidas son extraídas de patrones. Cada patrón es representado por un vector característico y el reconocimiento de cada patrón es usualmente realizado al particionar el espacio de la característica. Por otro lado, en el enfoque sintáctico, cada patrón es expresado como una composición de sus componentes, llamados sub-patrones. El reconocimiento de cada patrón es usualmente realizado al pasar la estructura del patrón de acuerdo a un conjunto de reglas.(Fu & Rosenfeld, 1976)

Algoritmos de procesamiento de imágenes

El reconocimiento de patrones y el procesamiento de imágenes se han desarrollado como disciplinas separadas, sin embargo, están muy relacionadas. El área de procesamiento de imágenes no solo consiste en la codificación, filtrado y mejoramiento, también se necesita el análisis y reconocimiento de imágenes. El área del reconocimiento de patrones incluye no solo las características de extracción y clasificación, sino también el pre-procesamiento y descripción de patrones.

En el caso particular de señales de 2D es necesario pre-procesar las señales con el fin de eliminar

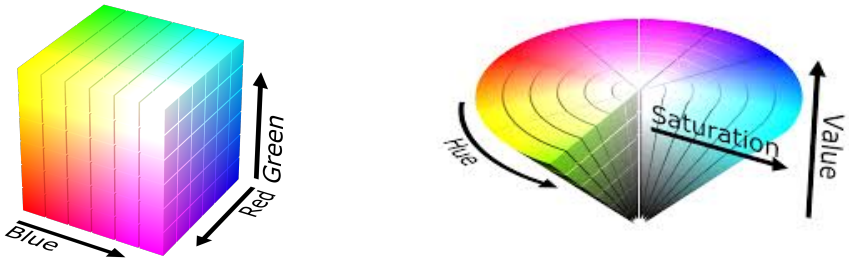
ruido, hacer transformaciones que mejoren la representatividad de los atributos, entro otros (Duda & Stork, 2001). Estos algoritmos pertenecen al área de procesamiento de imágenes, en el cual una imagen de entrada es transformada en una imagen de salida, con determinadas características especiales, donde se resalta o se atenúa información dependiendo de la aplicación. Por lo tanto, las tareas del procesamiento de imágenes comprenden la supresión de ruido, mejoramientos de contraste, eliminacioón de efectos no deseados como difuminación o distorsión, mapeos geométricos y transformaciones de color. Estos algoritmos facilitan el procesamiento posterior para el reconocimiento de patrones. (Moya, 2012)

Los algoritmos de procesamiento de imágenes se pueden clasificar en tres niveles: bajo, intermedio y alto.(Soviany, 2003), (Wu, 2013)

Los algoritmos de procesamiento de imágenes de bajo nivel tienen como tarea, extraer las primitivas fundamentales de una imagen para otros procesos, incluyendo detección de bordes, detección de esquinas, filtrado y morfología. Operan en toda la imagen, para generar un valor, un vector u otra imagen. Por el concepto de localidad espacial que tiene, donde se trabaja con píxeles individuales o en áreas de una imagen, ofrece facilidades para el paralelismo fino de datos, procesos de filtrado, convolución, generación de histogramas, suavizado y agudizado de imágenes, son ejemplos de procesamiento de bajo nivel.

Las operaciones de nivel intermedio producen estructuras de datos más compactas a partir de los píxeles de un objeto, como listas de una imagen de entrada. Debido a que estos procesos trabajan en segmentos de una imagen, estas operaciones ofrecen un grado intermedio de paralelismo, son más restrictivos al paralelizar los datos que las operaciones de bajo nivel. La transformada de Hough, el etiquetado de objetos y el análisis de movimiento, son ejemplos de estas operaciones. Existen dos aspectos importantes en este nivel: inferir la geometría e inferir el movimiento.

Las operaciones de alto nivel en procesamiento de imágenes se caracterizan por ser procesos donde se trabaja con estructuras de datos de entrada para generar, otras estructuras de datos que llevan a la toma de decisiones en una aplicación, es decir, infiere la semántica. Usualmente tienen patrones de acceso irregulares. Debido a estas propiedades, las operaciones de alto nivel ofrecen paralelismo grueso y es difícil de hacer paralelización de píxeles. Estimación de posición y reconocimiento de objetos son



(a) Representación RGB **(b)** Representación HSV

Figura 3.2: Representaciones de los espacios de color RGB y HSV

ejemplos de procesamiento de alto nivel.

Imágenes en color

Una imagen contiene uno o más canales que definen la intensidad o el color en una posición particular de píxel *I(m,n)*.

En una representación simple de una imagen de un canal (blanco y negro), cada posición de píxel contiene un solo valor numérico que presenta el nivel de la señal en el punto de la imagen. La conversión de este conjunto de números a una imagen se logra por medio de un mapeo de color. Un mapeo de color parea una sombra de color específica a cada nivel numérico en la imagen para asignar una representación visual del dato. El mapa de color más común es la escála de grises, que asigna todas las sombras de gris a negro (cero) hasta blanco (máximo) de acuerdo al nivel de la señal. La escala de grises es particularmente apropiada para la intensidad de imagen.

Adicional a las imágenes en escala de grises, tambien existen representaciones continuas de imágenes de color donde el espectro completo de color puede entenderse como un vector triple, con los componentes (R,G,B) que corresponden a los colores rojo, verde y azul, en cada posición de píxel, una representación de este espacio de color se observa en la figura 3.2a. En este, el color es representado como una combinación lineal de los colores básicos, la imagen es una representación que consiste de tres planos de dos dimensiones.

Existen otros sistemas de color, como (H,S,V) que corresponde a una representación del matiz (*Hue*), la saturación (*Saturation*) y valor o intensidad (*Value*) como se muestra en la figura 3.2b. En este sistema de color, la intesidad V del color es desacoplada de la información cromática, que esta contenida dentro de los componentes S y H. (Solomon & Breckon, 2011)

Segmentación de imágenes

La segmentación es el proceso donde una imagen es subdividida en sus regiones constituyentes ú objetos. El nivel de la subdivisión depende de la aplicación, esto es, la segmentación se detiene hasta encontrar los objetos de interés. La precisión de este proceso determina los resultados de los procesos posteriores como el rastreo de objetos.

Los algoritmos de segmentación de imágenes generalmente se basan en dos propiedades de la intesidad de valores: discontinuidad y similitud. Los algoritmos basados en la primera propiedad se basan en segmentar basadaos en cambios abruptos en la intensidad, tales como los contornos en una imágen. Aquellos algoritmos basados en similitudse basan en encontrar regiones que son similares deacuerdo a un conjunto de cirterios predefinidos. (Gonzalez & Woods, 2008).

3.2. Grafos

Un grafo es una representación abstracta que consiste en un conjunto de vértices, que representan objetos, y un conjunto de bordes que representan asociaciones entre pares de objetos. Un grafo también es conocido como red,los vértices como nodos; y los bordes como arcos, enlaces, lados o conexiones (Phillips, 2015).

Formalmente, la notación a utilzar es la siguiente: Un grafo G es un par $[G = (V,E)]$, donde $V(G)$ es un conjunto finito (vértices y nodos) y $E(G)$ denota el conjunto de pares no ordenados o lados del grafo. Típicamente se utiliza solo V y E cuando no existe ambigüedad. Además $v(G)$ y $e(G)$ indican el número de vértices y el número de bordes de G respectivamente. El número de vértices es el *orden* del grafo (Rosen, 2004).

La cardinalidad de V se denota por $|V|$ o cuando no existe ambigüedad por n. La cardinalidad de E se denota por $|E|$. Los bordes se identifican por los vértices que conectan; por ejemplo *(u,v)* indican un borde entre los vértices u y v. Los vecinos de un vertice $v \epsilon$ *V(G)* esta definido por $N_G(v) = u\epsilon V(G) : uv\epsilon E$ y también se conoce como *N(v)* cuando no existen ambigüedades.

Un grafo es multipartito si el conjunto de vértices en el grafo pueden ser divididos en subconjuntos no vacios, llamados partes, de tal forma que no existan cualquier par de vértices en un mismo grafo que tengan un borde conectándolos. Un grafo multipartito completo es un grafo multipartito tal que cualquier par de vértices que no estan en una misma parte tengan un borde conectándolos, por ejemplo en la figura 3.3 se muestra un grafo bipartito completo que cumple esta condición, conteniendo todos los bordes posibles. Dos vértices se dicen vértices disjuntos si no tienen un vértice en común, a un a un conjunto de bordes disjuntos de vértices pares una coincidencia (Sitton, 1996).

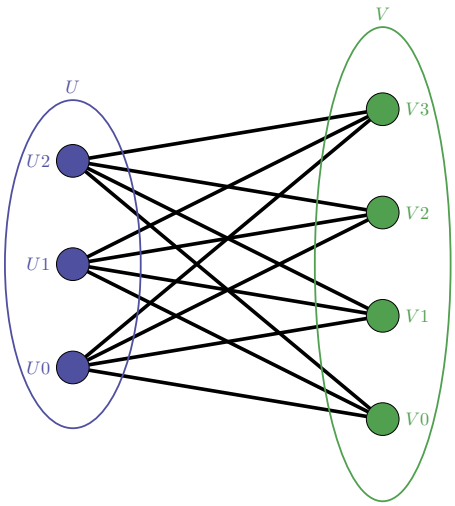


Figura 3.3: Grafo bipartito completo

Un grafo es k-partito si pueden ser particionado en k conjuntos independientes disjuntos. Un grafo k-partito es llamado multipartito, tipicamente solo cuando $k \geq 3$ y cuando $k = 2$ es llamado bipartito (Phillips, 2015).

Un grafo k-partito es balanceado si los conjuntos partitos difieren en el número de vértices como máximo en uno. A la vez, un grafo ponderado o pesado es un grafo en el que a cada borde se le asigna un peso numérico. Este peso representa por ejemplo un costo, distancia o una relación de similitud entre vértices (Kapanowski & Galuszka, 2015).

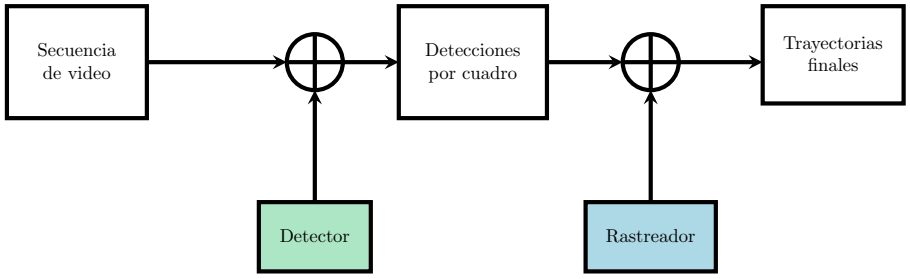


Figura 3.4: Una secuena de video se le aplica un detector a cada cuadro para encontrar regiones candidatas. Luego el rastreador realiza la asociación de datos para obtener las trayectorias finales.

3.3. Rastreo

El rastreo es comunmente dividido en dos pasos: la detección de objetos y la asociación de datos. Primero, los objetos son detectados en cada cuadro de una secuencia y luego, las detecciones son emparejadas para formar una trayectoria. Esto es conocido como el paradigma de rastreo por detección .

En la figura 3.4 es muestra un diagrama del paradigma de rastreo por detección, contiene dos componentes principales: el detector y el rastreador. Los detectores no son perfectos y frecuentemente producen falsas alarmas ú objetos no detectados, esto hace que el rastreo sea un proceso no trivial. Algunos de los retos más importantes son: (Taixe, 2016).

- Detecciones perdidas: Las oclusiones prolongadas están presentes en secuencias con alta densidad de objetos intereactuando, donde un detector puede perder el rastro de los objetos. En este caso, es díficil que el detector recupere las identidades de los objetos.
- Falsas alarmas: Esto corresponde a las regiones detectadas en la imagen que no contienen objetos de interés, creando falsos positivos y trayectorias no deseadas para los análisis posteriores.
- Apariencias similares: Una fuente de información para un rastreador es la apariencia, en el rastreo de jugadores de fútbol, donde existe gran similitud entre jugadores de un mismo equipo puede provocar intercambio de identidades.
- Agrupamiento y trayectorias impredecibles: En situaciones donde hay una alta densidad de jugadores en una región, como en los tiros de esquina y los movimientos propios de un jugador para evitar los jugadores contrincantes hace que detectores y rastreadores basados en la dinámica pierdan el rastro.

3.4. Procesamiento en paralelo

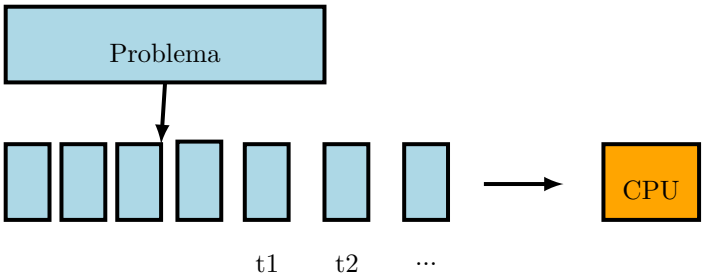


Figura 3.5: En computación secuencial un problema es resuelto con instrucciones que se procesan una a la vez por un CPU.

Tradicionalmente se han escrito los programas para ser ejecutados secuencialmente en un único procesador, donde un problema se resuelve con instrucciones que se ejecutan una a la vez y una después de la otra por el procesador, como se observa en la figura 3.5.

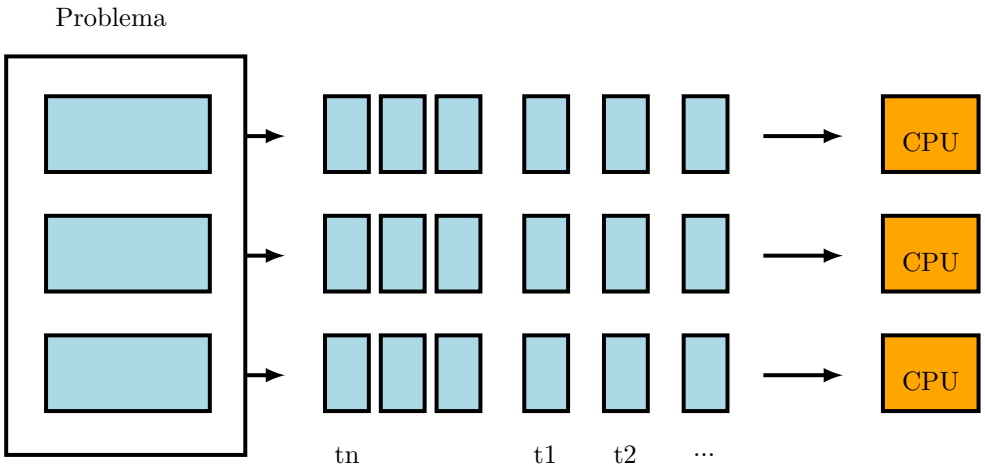


Figura 3.6: En computación paralela un problema es resulto con instrucciones que se procesan al mismo tiempo por varios CPUs.

El procesamiento en paralelo, es el uso simultáneo de múltiples recursos computacionales (procesadores y núcleos) para resolver un problema, figura 3.6. El problema es separado en múltiples partes para ser resuelto concurrentemente. A nivel de instrucciones, donde cada parte de un programa es ejecutada simultáneamente en múltiples unidades de procesamiento, o a nivel de datos, donde se distribuyen el conjunto de datos entre las unidades de procesamiento.

Existen dos formas de procesamiento paralelo(Pacheco, 2011):

- El procesamiento virtual utiliza una unidad de procesamiento donde el tiempo de ejecución es compartido por varios procesos, la calendarización por parte del sistema operativo, de estos procesos es tal que parece que estos procesos se realizan en forma simultánea.
- El procesamiento real utiliza varias unidades de procesamiento que ejecutan y comparten parte de un proceso o varios.

El sistema operativo es el elemento que permite generar un paralelismo virtual. El procesamiento paralelo real no necesariamente requiere de un sistema operativo. También se puede dar el caso en el cual existan ambos tipos de procesamiento paralelo, esto puede ocurrir, por ejemplo, cuando se tiene un sistema operativo corriendo en un sistema multiprocesador.(Tanenbaum, 2006)

Sistemas multiprocesador

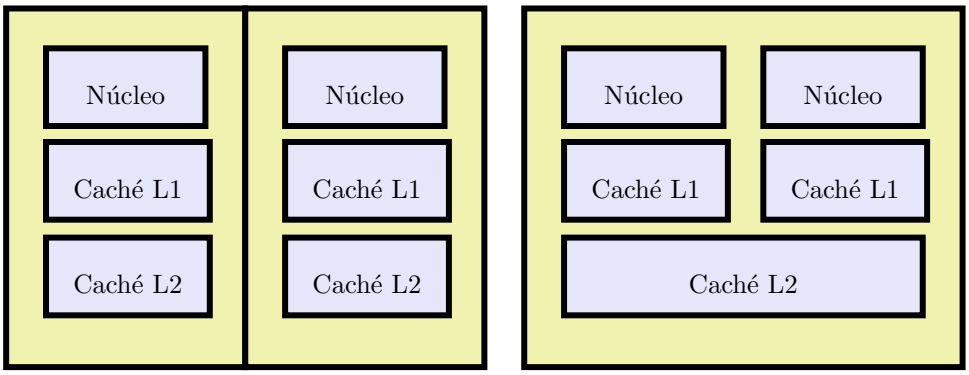


Figura 3.7: Sistema multiprocesador y multinúcleo

Un sistema multiprocesador, es una plataforma con más de un procesador que permiten que trabajen en paralelo y un sistema multinúcleo es una plataforma con un procesador, el cual tiene múltiples núcleos. En ambos casos, se trata de configuraciones donde se emplea el paralelismo como método, para aumentar el rendimiento efectivo pero mediante estrategias diferentes. En un sistema multinúcleo a nivel del procesador, hay un nivel de memoria que es compartida.(Mattson, Sanders & Massingill, 2005)

Una de las limitantes más serias de los sistemas multiprocesadores lo representan aquellas tareas que no se pueden ejecutar de forma paralela, llamadas secuenciales o seriales. Estas tareas representan verdaderos “cuellos de botella” para los sistemas multiprocesadores y tal como lo estable la Ley de

Amdahl limita bastante sus prestaciones. La Ley de Amdahl lo que establece, es que el aumento de velocidad de un sistema computacional con p, procesadores con respecto a un sistema con un único procesador, es menor o igual a la siguiente expresión:

$$\frac{1}{f + \frac{1-f}{p}} = \frac{p}{1 + f(p-1)} \tag{3.1}$$

en donde f es la fracción normalizada de las tareas que no pueden ser paralelizadas.

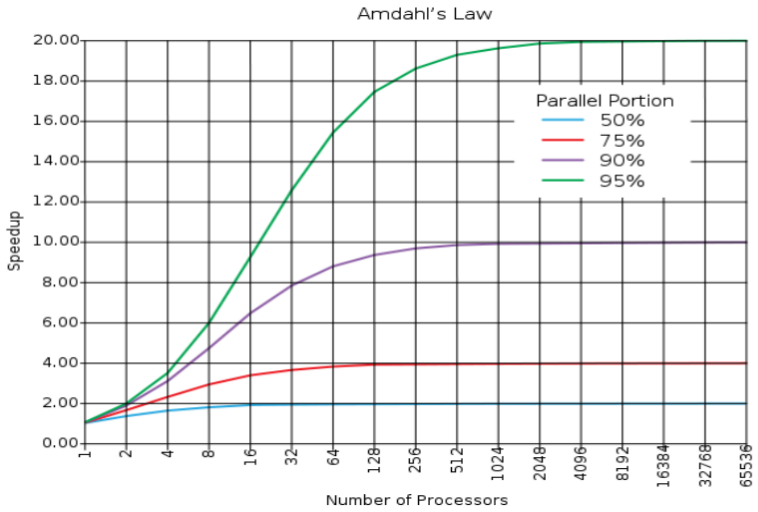


Figura 3.8: Aceleración según la Ley de Amdahl en función de la porción del programa que se puede paralelizar. Fuente: (Wire, 2019)

La ley de Amdahl, es un modelo que describe la relación entre la rapidez de la implementación paralela de un algoritmo y la implementación secuencial de este algoritmo. Según esta ley, la aceleración de un programa está limitada por la porción secuencial. Una limitación de esto es que supone que el tamaño del problema se mantiene constante al incrementar el número de procesadores. (Amdahl, 1967).

Un inconveniente de los sistemas multiprocesadores, en muchos casos, es el requisito de reescribir el software ya existentes para que pueda ejecutarse eficientemente en ellos. Dada la gran cantidad de software ya existente para sistemas uniprocesadores, es una tarea que requiere de un gran esfuerzo.

Programación en paralelo y definiciones

Existen varios formas de crear programas para sistemas de multiprocesador, la gran mayoría depende en la idea básica de dividir el trabajo entre los núcleos. Hay dos enfoques ampliamente usados,

paralelismo de tareas y paralelismo de datos. En el paralelismo de tareas, se dividen las tareas para la solución del problema entre los núcleos. En el paralelismo de datos, se dividen los datos utilizados en la solución del problema entre los núcleos y cada núcleo lleva a cabo operaciones similares en su parte de datos.(Michael McCool, 2012)

En un sistema de memoria compartida, los núcleos se coordinan y comunican al examinar y actualizar ubicaciones de memoria compartida. En un sistema de memoria distribuida, cada núcleo tiene su propia memoria que es de acceso privado, los núcleos deben comunicarse explícitamente por medio del envío de mensajes a través de una red. Cada núcleo puede ejecutar varios procesos e hilos.

En sistemas operativos se define, como un proceso, a un programa en ejecución que está compuesto por una secuencia de instrucciones, un espacio de direcciones de memoria, el estado de ejecución y un hilo de control. Los procesos pueden intercambiar información o sincronizar su funcionamiento a través de varios métodos de comunicación entre procesos.

Un hilo es la unidad de procesamiento más pequeña que puede ser planificada por un sistema operativo y permite a un proceso, realizar varias tareas de forma concurrente. A diferencia de los procesos, los hilos creados en un mismo contexto comparten los recursos directamente. Por estas características, es más rápido y tiene un menor costo cambiar de un hilo a otro dentro del mismo proceso.

MPI

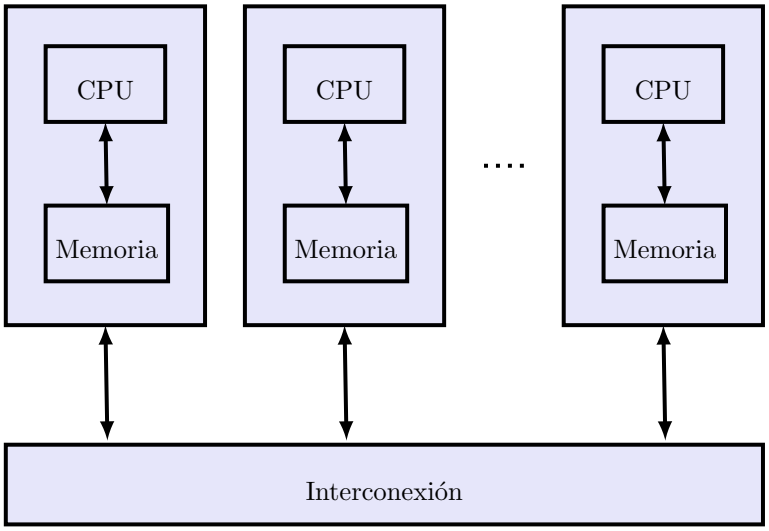


Figura 3.9: En un sistema con memoria distribuida los procesadores tienen acceso a la memoria local y a un espacio distribuido de memoria separada fisicamente.

MPI es un estándar de programación en paralelo para ambientes de memoria distribuida. Define una librería de funciones que pueden ser invocadas desde los lenguajes de programación C, C++ y Fortran. El objetivo de MPI es establecer un estándar portable, eficiente y flexible para el paso de mensajes. Incluye rutinas de sincronización de procesos, dispersión de datos entre procesos, manejo dinámico de procesos, comunicaciones bidireccionales y unidireccionales. Más de 115 rutinas están definidas en la versión MPI-1.

Existen varias implementaciones de MPI, las más comunes son LAM/MPI y MPICH. Soportan un amplio rango de computadoras paralelas, incluyendo clusters, computadoras NUMA y SMP. La gran mayoría de programas en MPI usan el modelo SIMD sin embargo es posible usar también el modelo MIMD. Muchos programas paralelos se pueden escribir con seis funciones básicas:MPI_INIT, MPI_Comm_Size, MPI_Comm_Rank, MPI_Send, MPI_Recv y MPI_Finalize.(Open-MPI-Project, 2015)

Conceptos básicos

La técnica central en MPI, es el paso de mensajes y la idea esencial es simple, un proceso envía un mensaje y otro lo recibe. Sin embargo, hay aspectos más complejos como el almacenamiento de los mensajes en el sistema, el aprovechamiento del tiempo del procesador mientras se realizan las co-

municaciones y la identificación de los mensajes. El enfoque de MPI, está basado en dos elementos principales, grupos de procesos y un contexto de comunicación.

Un grupo de proceso, es un conjunto de procesos envueltos en un cálculo computacional. En MPI todos los procesos se inician juntos, pero el programador los divide en subgrupos para controlar la interacción entre estos. El contexto de comunicaciones provee el mecanismo para agrupar los conjuntos de comunicaciones.

En un sistema de paso de mensajes, las comunicaciones deben estar etiquetados. Las etiquetas consisten en un identificador del proceso que envía, un identificador del proceso que recibe y un identificador del mensaje. En aplicaciones complejas, resulta inadecuado utilizar un sistema de este tipo, puesto que el programador usualmente no conoce los detalles y la reutilización de funciones pueden provocar conflictos en la identificación de los mensajes. MPI resuelve estos conflictos con la noción de contexto de comunicaciones. Cada elemento del etiquetado en MPI pertenece a un único contexto de comunicaciones que son creados dinámicamente, evita que los mensajes sufran de conflictos.(Forum, 2015)

OpenMP

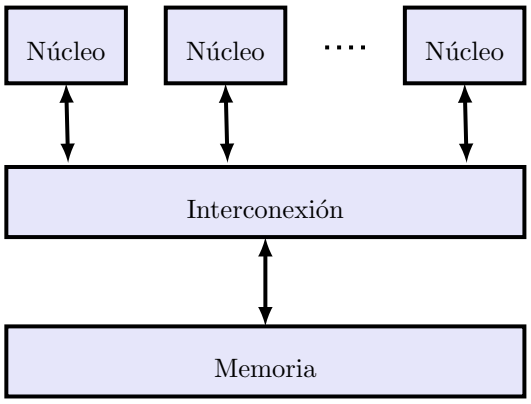


Figura 3.10: En un sistema de memoria compartida los procesadores comparten un espacio de memoria.

OpenMP es un API para programación en paralelo para sistemas con memoria compartida, la ejecución de este tipo de aplicaciones requiere de hilos. Está diseñado para sistemas, en donde, cada hilo puede tener acceso a toda la memoria disponible, cuando se programa con OpenMP, se ve al

sistema como un conjunto de núcleos que tienen acceso a la memoria principal, como en la figura 3.10.

Consiste en directivas de compilación, funciones y variables de ambiente. Las directivas se usan para inicializar las regiones paralelas, dividir el trabajo, sincronizar y manejar los atributos de los datos compartidos. La parte central es la paralelización de lazos. Los hilos se comunican utilizando variables compartidas, el uso inadecuado de variables compartidas origina carreras críticas. Para controlar las carreras críticas se usa sincronización al protegerse de los conflictos de datos.

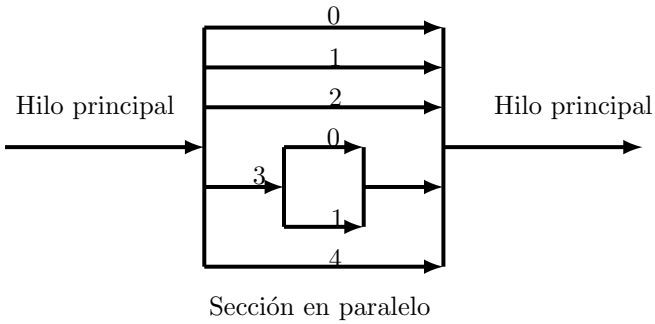


Figura 3.11: Modelo de paralelización OpenMP basado en la separación y unión de hilos.

Esta basado en un modelo de bifurcación y unión, un programa que utiliza OpenMP, inicia como un hilo principal como se muestra en la figura 3.11.

En un punto del programa donde la ejecución paralela es deseada, el programa inicializa hilos adicionales que se ejecutan paralelamente en una región del código, al finalizar esta región, los hilos esperan a que todos terminen y luego se unen de regreso al hilo maestro.

Este API fue diseñado en torno a dos conceptos claves: equivalencia secuencial y paralelismo incremental.

Un programa es secuencialmente equivalente cuando se producen los mismos resultados cuando se ejecuta usando un hilo o varios hilos. Paralelismo incremental se refiere al estilo de programación, en donde un programa evoluciona de su forma secuencial a su forma paralela, se trabaja por bloques de código para encontrar partes que se pueden ejecutar en paralelo.

Capítulo 4

Metodología

La investigación de la literatura profesional tratada en el capítulo 2, relacionada con el rastreo de jugadores en deportes colectivos da los fundamentos del estado actual del tema de investigación. Esta consulta de artículos públicados se obtienen de fuentes científicas como IEEE Explore®, ACM® y ScienceDirect® a partir de las descripciones de los algoritmos, experimentos y métricas encontradas en estas investigaciones científicas se establece la metodología idónea.

4.1. Ambiente de software

Para el cumplimiento del primer objetivo, es importante anotar que la plataforma de desarrollo consiste en una biblioteca en C++, de funciones y clases necesarias para la programación del algoritmo de rastreo de jugadores y que sean reutilizables en otras aplicaciones. En (Villalta, 2016) se propuso una biblioteca que aprovecha y se adapta a diferentes capacidades computacionales, como equipos portátiles o sistemas híbridos de alto desempeño. Por lo que la plataforma de desarrollo de esta investigación, mantendrá compatibilidad utilizando y extendiendo la funcionalidad de dicha biblioteca. La plataforma de rastreo automatizado consiste en un conjunto de herramientas como: un programa de anotación manual que genera los datos de referencia para la validación del funcionamiento, un traductor de anotaciones manuales de la base de datos ISSIA (Orazio, M.Leo, Mosca, P.Spagnolo & P.L.Mazzeo, 2009) y un programa para la validación de resultados que compará los resultados obtenidos con el algoritmo contra los datos anotados manualmente.

Tanto el algoritmo como la plataforma, se realizan utilizando un ciclo genérico de desarrollo (Intel Corporation, 2004) y la metodología propuesta en (Mackay, 2009), enfocada a bibliotecas, compuestas por las etapas de análisis, implementación, exactitud y optimización. Las bibliotecas o herramientas utilizadas para el desarrollo de los componentes anteriores son: opencv-2.4.13.5 para realizar el pro-

cesamiento de imágenes, yaml-cpp-0.6.0 para la creación de los archivos con los resultados de rastreo, openmpi-2.1.3 y boost-1.67.0 para la distribución y serialización de datos necesarios en la versión en paralelo del algoritmo.

En el PRIS-Lab se dispone de una herramienta de software llamada GT-Tool, para realizar anotaciones manuales por medio de una interfaz web, que valida los distintos bloques del sistema ACE. A pesar de la existencia de GT-Tool se desarrolla una aplicación tipo *stand-alone* que comparte funciones implementadas en el algoritmo con el fin de matener la compatibilidad de los formatos de salida. Se describe en detalle todos los componentes de la plataforma de rastreo en el capítulo 5.

4.2. Ambiente de hardware

Figura 4.1: Clúster Tará.

En esta sección se hace una descripción de la plataforma de ejecución. Debido a las características de los datos, es necesario que se ejecute el algoritmo en un ambiente de hardware de alto desempeño como un clúster.

Componentes de Hardware

Nodo	Cantidad	Procesador	Ram	HD	GPU
Maestro	1	Intel Xeon E5-2650	62 GB	500 GB	-
Procesamiento	4	Intel Xeon E5-2650	251 GB	500 GB	Tesla K20m
Almacenamiento	4	Intel Xeon E5-2650	62 GB	40 TB	-

Tabla 4.1: Especificaciones de hardware del clúster Tará

La descripción de los componentes generales de hardware del clúster se resume en el cuadro 4.1. El clúster HPC Tará incluye 4 nodos de procesamiento, cada uno con dos procesadores de 8 núcleos, donde cada núcleo es capaz de ejecutar 2 procesos/hilos concurrentes (128 procesos en total) y con dos tarjetas Nvidia Tesla K20m, es importante comprender la arquitectura base de los procesadores para establecer una relación óptima para la ejecución del algoritmo. Los 4 nodos de almacenamiento proveen un total de 160 TB. El nodo maestro funciona como enrutador, gestor y calendarizador de tareas. Todos estos 9 nodos cuentan con dos procesadores Intel Xeon E5-2650 operando a una velocidad de 2.60 GHz y 500GB de almacenamiento local.

Cada nodo dispone de 4 interfaces de red Gigabit Ethernet y 4 interfaces de red 10 Gigabit Ethernet. Los nodos están interconectados por medio de un conmutador Dell PowerConnect 8100 de 10 Gigabit Ethernet, en una topología tipo estrella, cada nodo esta conectado al conmutador utilizando 4 interfaces de red. El nodo maestro, al ser el enrutador del clúster, tiene reservada una conexión a la red telemática de la U.C.R..

4.3. Validación

Métricas

Rastreo

Dado que es necesario una evaluación para la comparación de los algoritmos, diferentes criterios se han usado en distintos conjuntos de datos para medir las cualidades cuantitativas y cualitativas de los mismos. Para ello es necesario también crear un conjunto de datos de referencia (conocido en inglés como *ground truth*), comúnmente anotando de forma manual las posiciones de los jugadores con alguna herramienta de interface gráfica. En (Manafifard, Ebadi & Moghaddam, 2017) se hace un extenso estudio sobre las métricas utilizadas en el rastreo de jugadores de fútbol, en esta revisión se

indica la falta de consenso en la utilización de una sola métrica, lo que dificulta la comparación entre los distintos algoritmos; así como también de la fuente de datos utilizadas por los distintos autores, donde la gran mayoría de conjuntos de datos no se encuentran disponibles para la comunidad científica.

A pesar de lo anterior, en las investigaciones de rastreo de objetos, un campo más amplio, se utiliza en su mayoría un conjunto de métricas y notaciones que permiten la evaluación de los algoritmos, las cuales se mencionan a continuación:

- GT: Se refiere a los datos anotados manualmente.
- SUT: Sistema bajo valoración.
- FP: Falso positivo, cuando un objeto esta presente en el SUT, pero no en el GT (falsa alarma).
- FN: Falso negativo, cuand un objeto esta presente en el GT, pero no en el SUT (Fallo de detección).
- TP: Verdadero positivo, un objeto esta presente en el GT y en el SUT (Detección correcta).
- TN: Un elemento no esta presente en el GT y en el SUT.
- CGT: Datos completos anotados manualmente.
- FPR: Métrica que indica el correcto rechazo de los falsos positivos por el sistema $FPR = FP/(FP + TN)$.
- FAR: Métrica que indica la similitud que un objeto detectado es correctamente reportado $FAR = FP/(TP + TN)$.
- DR: Métrica del porcentaje de objetos verdaderos que son detectados por el SUT $DR = TP/(TP + FN)$.
- FNR: Medida que indica la similitud que un objeto será perdido dado un número total de objetos reales $FNR = FN/(TP + FN)$.
- TNR: Medida que indica la similitud a una respuesta negativa dada entre el total de detecciones negativas $TNR = TN/(TN + FP)$.
- $Accuracy$: Medidad del desempeño actual del sistema con respecto a la correcta detección y el correcto rechazo de objetos. $Accuracy = (TP + TN)/CGT$.

- *Precision*: Fracción de los objetos detectados que son correctos, es el número de positivos verdaderos realtivo a la suma de los positivos verdadeos y los falsos positivos $Precision = TP/(TP+FP)$.
- *Recall*: Fracción de los objetos que fueron detectados correctamente entre todos los objetos que debían ser detectados $Recall = TP/(TP + FN)$
- *F-Measure*: Estimado de la exactitud del SUT $F1 = 2*(Precision*Recall)/(Precision+Recall)$

Cabe destacar que el rastreo múltiple de objetos, es un área muy activa de investigación donde año con año se proponen nuevos algoritmos. Sin embargo, tampoco hay un consenso sobre las métricas de desempeño que se utilizan en experimentos, esto hace que la comparación de los algoritmos sea difícil, dado que dependiendo de la métrica usada, los resultados del rastreo pueden estar sesgados a ciertos aspectos particulares del rastreo (Cehovin, Leonardis & Kristan, 2016). Además, la disponibilidad de los datos con los cuales se realizan los experimentos de rastreo de jugadores, no son de fácil y libre acceso. A pesar de esto, para el rastreo de múltiples objetos, existen dos métricas populares para medir la efectividad de los algorítmos: *Multiple Object Tracking Precision* (MOPT) y *Multiple Object Tracking Accuracy* (MOTA) (Bernardin & Stiefelhagen, 2008). Formalmente están dadas por:

$$MOTP = \frac{\sum_{i=1}^{M}\sum_{t=1}^{N}\phi_{i,t}}{\sum_{t=1}^{N} M_t} \tag{4.1}$$

$$MOTA = 1 - \frac{\sum_{t=1}^{N}(c_m MI_t + c_f FP_t + c_s SW_t)}{\sum_{t=1}^{N} N_t^G} \tag{4.2}$$

MOTP se define como el promedio de cubertura de todos los objetos en todos los cuadros, donde M denota el numero de diferentes objetos en toda la secuencia y M_t es el numero visible de objetos en el cuadro t. La métrica MOTA considera tres componentes que se toman para la precisión del algoritmo: número de pérdidas, número de falsas alarmas y número de identidades intercambiadas. El número de pérdidas esta denotado por MI_t, FP_t son el número de detecciones incorrectas, SW_t es el número de identidades intercambiadas, c_m, c_f y c_s son pesos constantes y N_t^G es el número de objetos anotados en el tiempo t. Estas serán consideradas como las métricas para la evaluación de los experimentos de esta investigación.

El programa de validación, implementa las fórmulas de las métricas presentadas y genera el conjunto de resultados a partir de la comparación de los datos anotados manualmente con los datos

generados por el algoritmo.

Aceleración y Eficiencia

El objetivo principal de paralelizar un algoritmo es mejorar su desempeño. Usualmente se pretende dividir el trabajo de forma equitativa entre los p procesadores asignando un proceso o hilo a cada procesador. Si se considera al tiempo de ejecución de la versión secuencial del algoritmo como $T_{secuencial}$ y al tiempo de ejecución de la versión paralela como $T_{paralelo}$, el resultado ideal al paralelizar el algoritmo sería cuando $T_{paralelo} = T_{secuencial}/p$, esto se conoce como aceleración lineal. Sin embargo en la práctica por motivos del software y hardware se introduce tiempo correspondiente a la creación de los hilos/procesos, accesos a memoria y a las comunicaciones, así como también existen secciones de los algoritmos que no se pueden paralelizar, por lo tanto el valor ideal no se puede alcanzar (Pacheco, 2011).

Se define como aceleración ó *speedup* de un programa paralelo como:

$$S = \frac{T_{secuencial}}{T_{paralelo}} \tag{4.3}$$

La eficiencia de un algoritmo en paralelo se define como:

$$E = \frac{S}{p} = \frac{T_{secuencia}}{p * T_{paralelo}} \tag{4.4}$$

Tanto la aceleración como la eficiencia, dadas por las ecuaciones 4.3 y 4.3, son métricas utilizadas para cuantificar el aprovechamiento al paralelizar un algoritmo.

Conjunto de datos

La metodología propuesta por (Petsas & Kaimakis, 2016) consiste en generar los videos y datos utilizados como "*ground-truth*" desde un modelo simulado que usa el motor de juegos Unity, se considera esta forma como una alternativa para generar las trayectorias de referencia. En este caso se puede utilizar la métrica creada en (Li, Dore & Orwell, 2005).

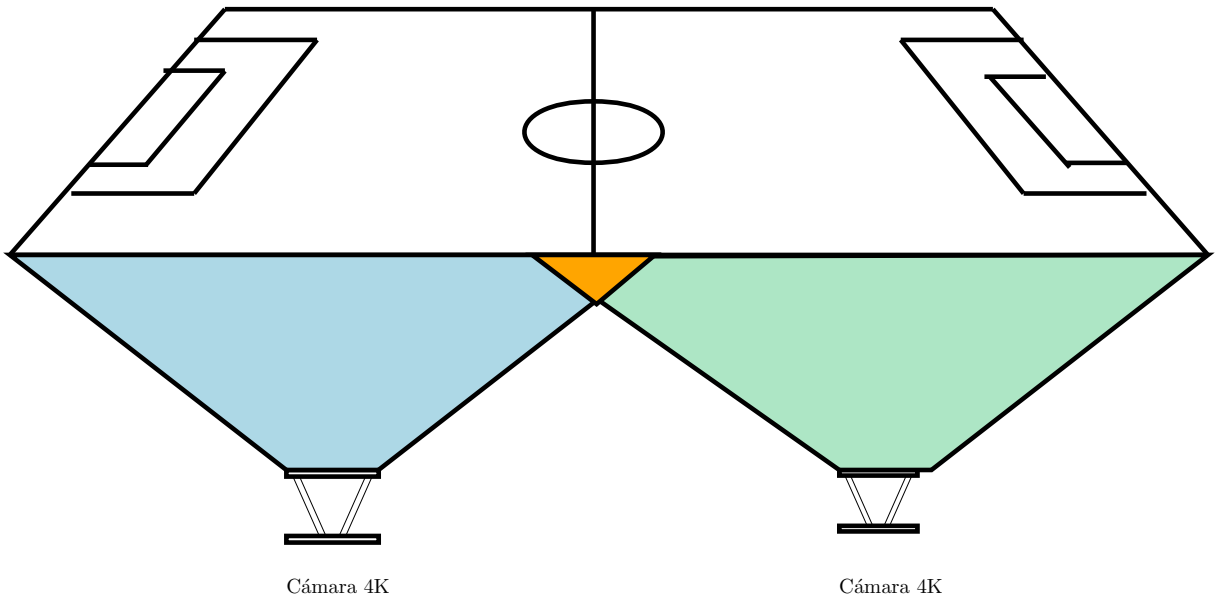


Figura 4.2: Posicionamiento de cámaras para la captura lateral del campo.

Figura 4.3: Imágen panorámica de entrada al rastreador

Para la generación de datos se utiliza la metodología propuesta por (Baysal & Duygulu, 2016). El conjunto de datos consiste en una secuencia de video producto de dos vistas sincronizadas como se muestra en la figura 4.2, estas vistas se superponen en una región para facilitar la unión de estas vistas en una sola vista como se observa en la figura 4.3. La longitud de cada secuencia de videos es variable y puede ir en un rango de 2 minutos a 45 minutos con una dimensión de 2k para cada cámara a una frecuencia de 30 cuadros por segundo. La secuencia resultante cubre el campo completo; así como tambíen a los jugadores y árbitros.

En el cuadro 4.2 se muestran los parámetros de configuración para la grabación de los videos, estos videos se graban por un equipo de miembros del PRIS-Lab en el estadio ecológico de las instalaciones deportivas de la U.C.R. En la figura 4.5 se muestran las condiciones en las que se realizan grabaciones en dicho estadio, se observa la ubicación y perspectivas de las cámaras con respecto a la cancha, así como también las condiciones de iluminación y ambientales. Para estas sesiones no se recomienda la

Tabla 4.2: Configuración de cámaras PXW-Z100

Parámetro	Valor
Format	Acquisition Metadata
Muxing mode	Ancillary data / RDD 18
Frame rate	29.970 fps
FocusPositionFromImagePlane_FirstFrame	65543.992 m
MacroSetting_FirstFrame	On
LensZoom35mmStillCameraEquivalent_FirstF	-4295007232.000 m
LensZoomActualFocalLength_FirstFrame	-536897152.000 m
OpticalExtenderMagnification_FirstFrame	100.00 %
AutoFocusSensingAreaSetting_FirstFrame	Manual
NeutralDensityFilterWheelSetting_FirstFr	Clear
ImageSensorDimensionEffectiveWidth_First	5.161 mm
ImageSensorDimensionEffectiveHeight_Firs	2.722 mm
CaptureFrameRate_FirstFrame	29.970 fps
ImageSensorReadoutMode_FirstFrame	Progressive frame
ShutterSpeed_Time_FirstFrame	1/250 s
CameraMasterGainAdjustment_FirstFrame	0.00 dB
ElectricalExtenderMagnification_FirstFra	100.00 %
AutoWhiteBalanceMode_FirstFrame	Automatic
WhiteBalance_FirstFrame	6799 K
CameraMasterBlackLevel_FirstFrame	2.00 %
ExposureIndexofPhotoMeter_FirstFrame	1000

Figura 4.4: Cámara Sony PXW-Z100.


(a) Perspectiva diagonal de cámaras


(b) Perspectiva central de cámaras


(c) Ubicación de cámaras

Figura 4.5: Sesión de adquisición de videos estadio Ecológico U.C.R.

perspectiva diagonal que se muestra en la figura debido a la rotación excesiva necesaria para juntar las vistas de las cámaras.

Las cámaras que se utilizan en las sesiones de captura son Sony modelo PXW-Z100 (figura 4.4), un detalle importante es la posibilidad de sincronizar los relojes de las cámaras por medio de la terminal *TC Link*, una de las cámaras se configura como generador de la señal de sincronización mientras la otra cámara se configura como recepctor de esta señal. Este modelo de cámara usa sensores CMOS Exmor capaces de resolución 4K (4096 x 2160) a 60 FPS ó 50 FPS. El formato de grabación es XAVC, utiliza un muestreo de imágenes de 4:2:2 10 bits a 4K 60 FPS, por lo que es posible tener una velocidad de bits de 500 Mbps ó 600 Mbps. También se pueden seleccionar las resoluciones QFHD (3840 x 2160) y HD (1920 x 1080).

En los cuadros 4.3 y 4.4 se muestran los videos grabados bajos las condiciones mencionadas anteriormente, además de mostrarse otros videos disponibles para la validación de los algoritmos de análisis de fútbol en el PRIS-Lab, como se denota previamente, todos aquellos videos que provienen de una fuente de transmisión de televisión no se toman en cuenta para este trabajo. En este cuadro también se indican videos grabados de partidos de la primera división profesional de fútbol de Costa Rica, de estos partidos el encuentro entre UCR-Cartagines es grabado a nivel de la cancha, debido a esto la cantidad de oclusiones es mucho mayor que las grabaciones realizadas en planos superiores y más alejados de la cancha de fútbol, por lo que se descarta.

Tabla 4.3: Videos grabados por el PRIS-Lab

Partido	Duracion	Calidad	Estadio	Fecha
Amistoso equipo femenino UCR	60 min	2 camaras 4K @ 30 fps	Ecológico	Domingo 10 de junio 2018
Amistoso equipo masculino UCR-UACA	73 min	2 camaras 4K @ 30 fps	Ecológico	Miercoles 8 agosto 2018
Amistoso equipo masculino UCR-UACA	41 min	2 camaras 4K @ 30 fps	Ecológico	Miercoles 4 de abril 2018
Intersedes UCR: Puntarenas – Rodrigo Facio	40 min	2 camaras 4K @ 30 fps	Ecológico	Miercoles 5 de setiembre 2018
Primer division UCR-Alajuelense	20 min			
Prmier division UCR-Santos	48 min	2 camara HD @ 30 fps	Ecológico	16-08-2015
Primer division UCR-Herediano	38 min	2 camara HD @ 30 fps	Nacional	29-08-2015
Primer division UCR-Cartagines	50 min	2 camara HD @ 30 fps	Cartago	13-09-2015

4.4. Experimentos

En esta sección se realiza una descripción de las condiciones de los experimentos realizados para validar la hipótesis planteada en la sección 1.

Rastreo

Efecto de la resolución

La habilidad de obtener calidad visual y precisión de los objetos rastreados es importante para un sistema de rastreo de jugadores, esto requiere una resolución apropiada de los datos de entrada para tener más detalles para el rastreador. Una baja calidad o una resolución pequeña del video, ruido del sistema, objetos pequeños y otros factores generan una menor precisión en los resultados del rastreador, bajo estas circunstancias es más difícil identificar a los jugadores y sus trayectorias (Habibi, Sulistyaningrum & Setiyono, 2017; Figueroa, Leite & Barros, 2004).

Dado lo anterior, es importante cuantificar la ganancia en el desempeño del rastreador en función de la resolución de los datos de entrada, para esto se determina un conjunto de escenas de longitud fija de 2 minutos y se cambia la resolución del video en 2x4K,4K y HD. Se utiliza la aplicación ffmpeg para redimensionar el video de origen, cuya resolución es de 8K, a las resoluciones menores.

Efecto del tamaño del grafo multipartito

El tamaño k del grafo multipartito determina el numero de cuadros de video en los cuales las trayectorias existentes pueden encontrar sus asociaciones y por lo tanto es importante analizar su efecto en los resultados de rastreo. Intiutivamente, al utilizar más cuadros en cuenta debería ser más factible resolver las detecciones inapropiadas y las oclusiones.

Tabla 4.4: Videos de transmisión televisiva

Partido	Duracion	Calidad
FWC2014		
Final GER vs ARG	210 min	1 camara HD @ 30 fps
BRA vs COL	140 Min	1 camara HD @ 30 fps
FRA vs GER	210 min	1 camara HD @ 30 fps
GER vs ALG	210 min	1 camara HD @ 30 fps
CMR vs BRA	140 Min	1 camara HD @ 30 fps
CMR CRO	140 Min	1 camara HD @ 30 fps
CRO MEX	140 Min	1 camara HD @ 30 fps
AUS ESP	128 min	1 camara HD @ 30 fps
AUS NED	140 Min	1 camara HD @ 30 fps
CHI AUS	140 Min	1 camara HD @ 30 fps
ESP CHI	140 Min	1 camara HD @ 30 fps
ESP NED	140 Min	1 camara HD @ 30 fps
COL GRE	140 Min	1 camara HD @ 30 fps
GRE CIV	140 Min	1 camara HD @ 30 fps
JPN GRE	125 Min	1 camara HD @ 30 fps
CRC ENG	140 Min	1 camara HD @ 30 fps
ITA URU	140 Min	1 camara HD @ 30 fps
URU CRC	140 Min	1 camara HD @ 30 fps
URU ENG	130 Min	1 camara HD @ 30 fps
ECU FRA	140 Min	1 camara HD @ 30 fps
HON SUI	140 Min	1 camara HD @ 30 fps
ARG BIH	140 Min	1 camara HD @ 30 fps
ARG IRN	130 Min	1 camara HD @ 30 fps
BIH IRN	135 Min	1 camara HD @ 30 fps
NGA ARG	135 Min	1 camara HD @ 30 fps
GER GHA	140 Min	1 camara HD @ 30 fps
GER POR	140 Min	1 camara HD @ 30 fps
POR GHA	140 Min	1 camara HD @ 30 fps
USA GER	140 Min	1 camara HD @ 30 fps
ALG RUS	140 Min	1 camara HD @ 30 fps
BEL ALG	140 Min	1 camara HD @ 30 fps
BEL RUS	140 Min	1 camara HD @ 30 fps
RUS KOR	140 Min	1 camara HD @ 30 fps

Para estudiar el efecto del tamaño k en el desempeño del rastreador se conduce un experimento con $k = 2, 5, 10, 15, 20, 30$ para una misma escena de una longitud de 2 minutos y resolución fija 4K.

Velocidad

El costo computacional de los algoritmos de rastreo es un aspecto relevante, dado que no sólo se requiere obtener las trayectorias de los objetos de interés, sino también es necesario obtenerlos en un tiempo razonable en especial si se tiene como datos de entrada un video de alta resolución. Para esto se pueden usar técnicas de reconocimiento de patrones al modificar el espacio de descriptores utilizando un conjunto de características que son más rápidas de obtener y que todavía preserven información útil para la discriminación en la toma de decisiones. Otra forma es la utilización de técnicas de paralelización de estos algoritmos, por lo tanto es importante medir los beneficios obtenidos de este tipo de enfoque.

Para estudiar estos beneficios se conduce un experimento donde se ejecuta el algoritmo de rastreo para un video con una longitud fija en las resoluciones de video 2x4K,4K y HD. Además se estudia la aceleración producto de las técnicas de paralelización con N hilos, donde $N = 1, 2, 4, 8, 16, 32, 64, 128$.

Capítulo 5

Plataforma de rastreo automatizado de jugadores

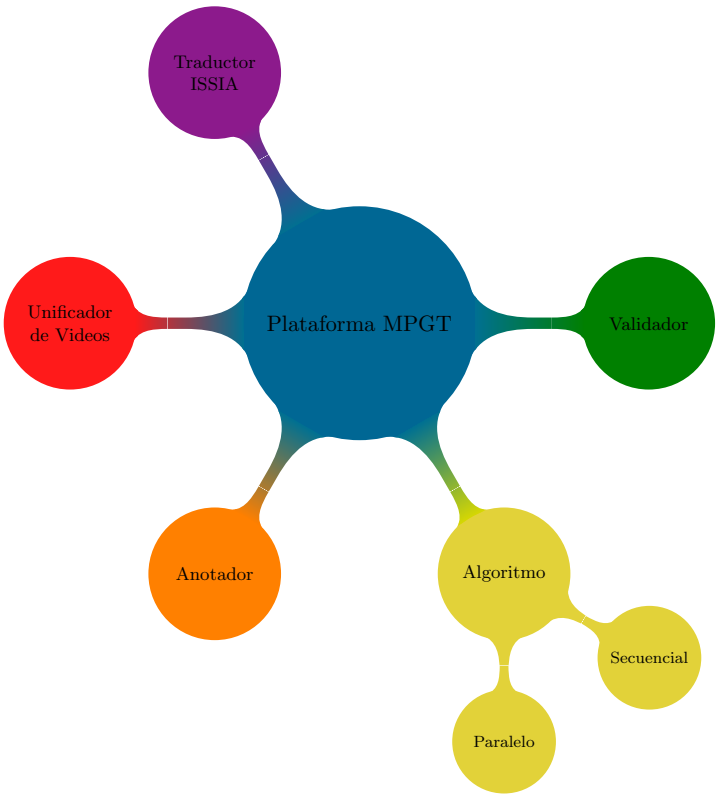


Figura 5.1: Elementos de la plataforma de rastreo automático

Como parte de los objetivos específicos del presente trabajo, la creación de una plataforma compuesta por un conjunto de aplicaciones es necesario para que acompañen y complementen el agoritmo de rastreo automatizado.

En la figura 5.1 se muestran los componentes que conforman la plataforma de rastreo automatizado de jugadores MPGT. Estos elementos funcionan como aplicaciones independientes, sin embargo

comparten funcionalidades y dependencias entre ellos. La implementación de estos componentes es en el lenguaje de programación C/C++ y utiliza las siguientes bibliotecas/herramientas:

- GCC 4.9.2
- OpenMPI 2.1.3
- OpenCV 2.4.13.5
- Boost 1.67.0
- Yaml-CPP 0.6.0

A continuación se hace una descripción del funcionamiento de estos componentes.

5.1. Algoritmo secuencial

El componente fundamental de la plataforma es el algoritmo de rastreo automatizado. Este algoritmo integra las etapas de segmentación espacial y rastreo de la arquitectura ACE, como se observa en la figura 1.1. Bajo el marco de un sistema de reconocimiento de patrones, la etapa de sensado es realizada por las cámaras 4K y el algoritmo se encarga de realizar las etapas de pre-procesamiento, segmentación, extracción de las características y clasificación.

Para la etapa de pre-procesamiento y segmentado se utiliza el algoritmo, como una versión modificada, propuesto en (F. Siles & Saborío, 2015) de segmentación espacial adaptado para propósitos de rastreo, que incluye las siguientes funciones:

1. Conversión del espacio de color de RGB a HSV.
2. Varianza local espacial normalizada para los canales de valor y matiz.
3. Identificación de las regiones verdes.
4. Detección de bordes.
5. Eliminación de las líneas de campo con la transformada de Hough.
6. Operaciones de transformación morfológicas de erosión y dilatación.

7. Descarte de regiones espurias y filtrado objetos usando un criterio de circularidad y tamaño.

8. Creación de los modelos de objetos dentro de un grafo multipartito.

Este conjunto de funciones de segmentación y filtrado se observa en la figura 5.2, donde se empieza con una imagen fuente y es procesada para obtener los contornos de los jugadores, que a la vez son la entrada de la etapa de rastreo que produce las ubicaciónes correspondientes a los objetos que representan a jugadores de fútbol.

Cuando el modelo del objeto es creado se extrae el vector de características compuesto por información de la forma, color, dirección y posición, para ser luego asignado como un nodo en el grafo correspondiente. Una vez poblado el grafo el algoritmo de rastreo realiza las siguientse funciones:

1. Filtrado de los bordes del grafo multipartito basado en la relación de similitud entre nodos que incluye la comparación de los siguientes descriptores estáticos y dinámicos: información cromática, posición, dirección, forma, área y velocidad.

2. Construcción de una matriz de coincidencias

3. Análisis hacia adelante de los bordes para la detección de eventos de unión/oclusión.

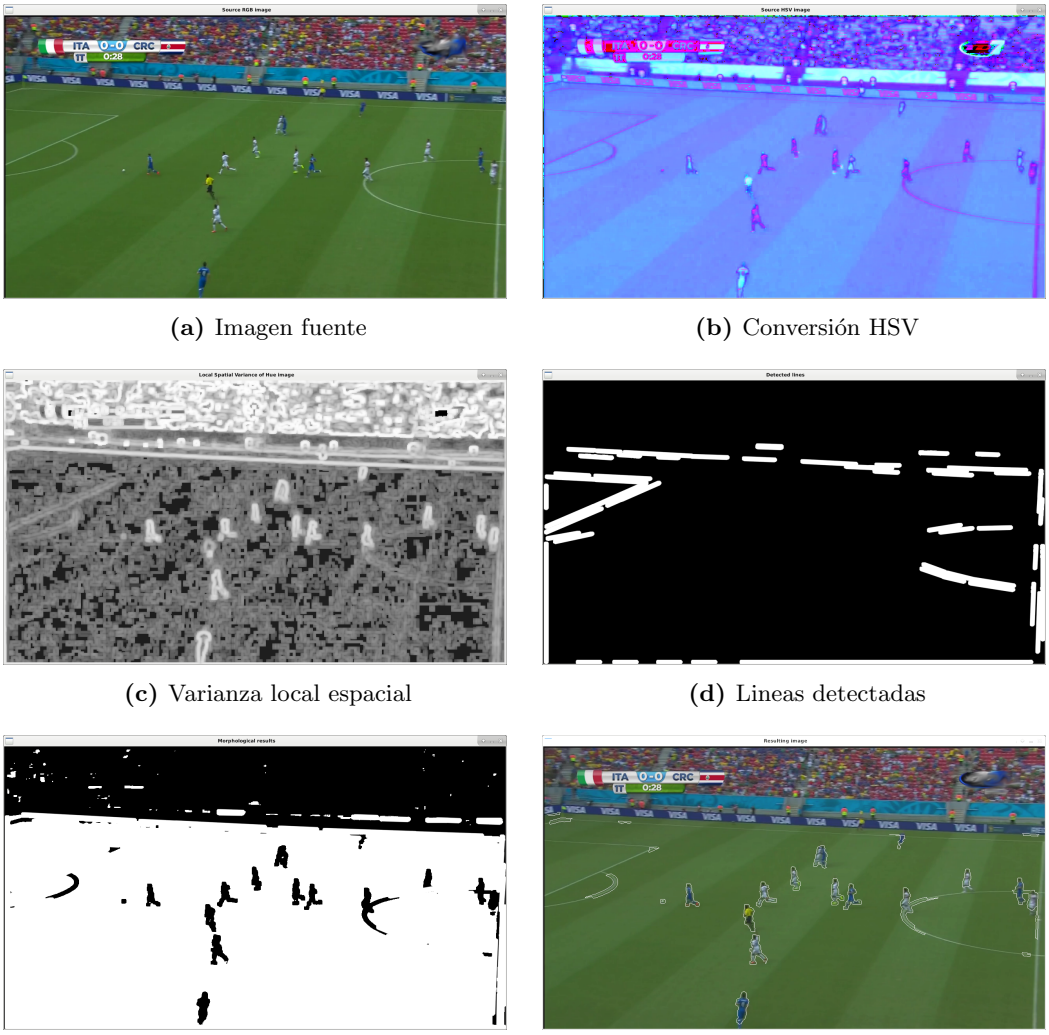
4. Análisis hacia atrás para la detección de eventos de separación.

5. Propagación de identidades.

De esta forma se obtiene un grafo como se muestra en la figura 5.3 que describe la relación de eventos existentes en un proceso de rastreo para un conjunto de cuadros dados, en esta figura se observa una secuencia de tres imágenes con su grafo multipartito.

El pseudo algoritmo que corresponde a la versión secuencial de los pasos mencionados anteriormente es el algoritmo 1 y se describe a continuación más formalmente. Sea un video V con v cuadros y G un grafo multipartito con k grafos, cada uno con n nodos. Donde f_n corresponde al número del cuadro procesado del video V, G_f es el primer grafo y G_t el último grafo de un grafo multipartito.

Como se puede observar en el algoritmo 1, el video V es procesado en grupos de cuadros (por lotes) iguales a los k cuadros para cada *paso*, el cual a la vez corresponde al primer cuadro de cada



(a) Imagen fuente **(b)** Conversión HSV

(c) Varianza local espacial **(d)** Lineas detectadas

(e) Regiones homogeneas **(f)** Contornos resultantes

Figura 5.2: Resultados de las etapas de la segmentación de jugadores

grupo, mientras $paso < v$. El cuadro f_n es leido de V y se le aplica la función de segmentación σ para generar los contornos C, luego cada contorno $c_i \in C$ es usado para generar el modelo del nodo N_j para el grafo G_i. Una vez que cada grafo multipartito ha sido poblado con k-grafos, la función de rastreo τ genera los T puntos de rastreo, los cuales son ubicaciones 2D en el cuadro como cordenadas (x,y). Para propagar las identidades entre grafos multipartitos consecutivos una función de correspondencia bipartita ζ es aplicada entre los grafos adyacentes.

(a) Primer imagen **(b)** Imagen intermedia **(c)** Última imagen

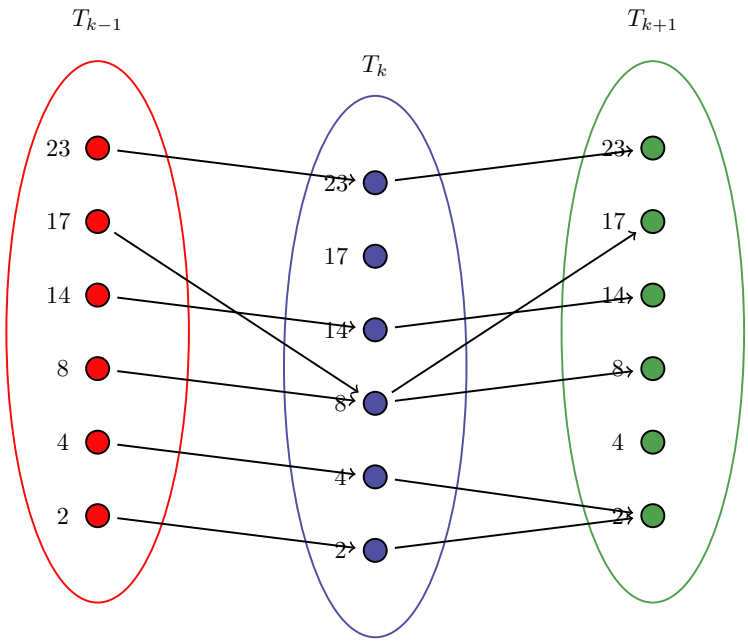


Figura 5.3: Secuencia de eventos de oclusión y rastreo con su grafo multiparito correspondiente.

Algorithm 1 Algoritmo secuencia de rastreo de jugadores con grafos multiparitos

Entrada V Video
Salida T Puntos de rastreo

procedure PROCESO POR LOTES
  paso $\leftarrow k$
  for $paso < v$ **do**
    procedure SEGMENTACION
      Inicializar G
      for $i < k$ **do**
        $f_n \leftarrow i + paso$ ▷ Actualizar el primer cuadro del grafo multipartito
        $C \leftarrow \sigma(f_n)$ ▷ Generar contornos al aplicar función de segmentación
        for $j < n$ **do**
          Generar modelo N_j ▷ Donde N es un nodo
          $G_i[j] \leftarrow N_j$ ▷ Guardar el modelo del nodo es su correspondiente grafo
        end for
      end for
    end procedure
    procedure RASTREO
      if $paso \neq f_f$ **then**
        ς ▷ Aplicar la funcion de comparación bipartita entre el grafo multipartito anterior y el grao multipartito actual
      end if
      $\tau(G)$ ▷ Aplicar función de rastreo
      $G_f \leftarrow G_l$
    end procedure
  end for
end procedure

5.2. Algoritmo en paralelo

Dada la configuración de cámaras que se muestra en la figura 4.2, dos cámaras 4K, que producen los datos de entrada al sistema y debido a la resolución UHD de 3840 x 2160 píxeles, a una razón de 60 Hz generando 995328000 de píxeles por segundo, es deseable un algoritmo capaz de procesar esta cantidad de información en una plataforma de hardware con los recursos suficientes para lograr resultados en tiempo real. Por lo tanto se desarrolla una versión en paralelo del algoritmo de rastreo automatizado.

A pesar del comportamiento secuencial que exhibe naturalmente el algoritmo 1, es posible traslapar sus dos procedimientos principales con un esquema consumidor-productor donde los procesos pares realizan la función de segmentación σ y generan un grafo multipartito que los procesos impares utilizan en la función de rastreo τ. Esta función de rastreo es realizada utilizando un patrón de comunicaciones de envio-recepción para hacer la propagación de las identidades entre procesos impares, los patrones de comunicaciones utilizados para paralelizar el algoritmo se muestran en la figura 5.4. Los pasos que

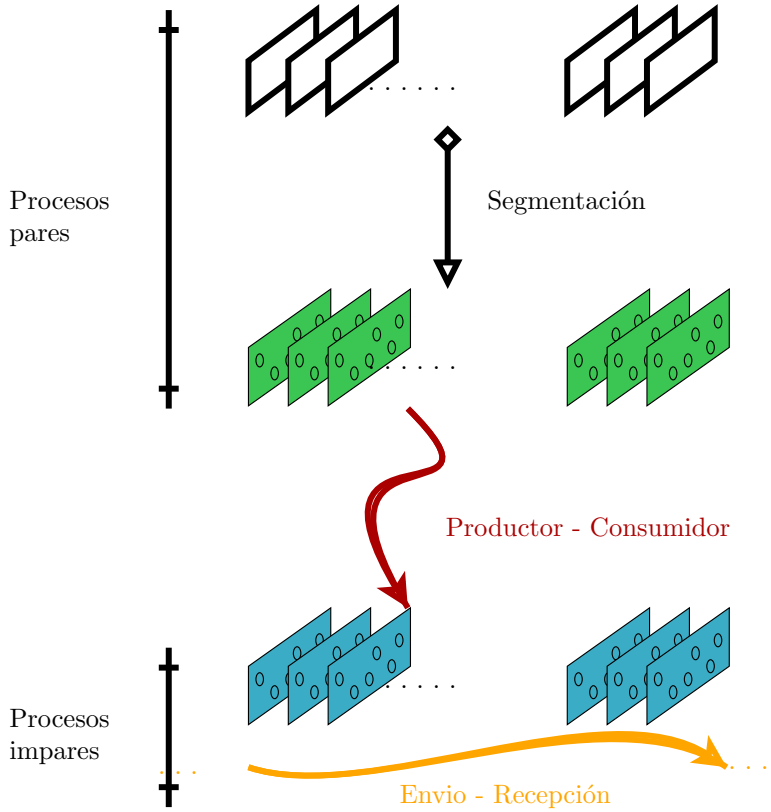


Figura 5.4: Patrones de comunicación y distribución de la tarea de segmentación en procesos pares y la tarea de rastreo en procesos impares de la versión en paralelo del MPGT.

realizan la función de segmentación σ y la función de rastreo τ son los mismos que la versión secuencial del algoritmo, sin embargo estas funciones se paralelizan utilizando el paradigma de OpenMP.

Siguiendo las misma notación asumida para el algoritmo 1, se asume que existen p procesos con su correspondiente identificación p_{id}, donde μ es el máximo número de iteraciones para esos procesos, f_a es el cuadro de inicio y f_o es el cuadro de parada para cada proceso, f_0 es el primer cuadro del video V, donde G_f y G_l son el primer y último grafo para el lote del proceso p_{id}. El algoritmo 2 describe más formalmente la versión en paralelo del algoritmo de rastreo, entre las lineas 2 y 24 esta contenido el procedimiento de segmentación y entre las lineas 25 y 75 el procedimiento de rastreo. Este algoritmo procesa la información por lotes, de las lineas 4 a 9 se muestra como se determina la distribución de cuadros para cada proceso y su correspondientes posiciones de inicio y final. Entre las lineas 12

Algorithm 2 Algoritmo paralelo de rastreo de jugadores con grafos multipartitos

Entrada V Video
Salida T puntos de rastreo

1: $\mu \leftarrow v/((p/2) * k)$
2: **procedure** PROCESAMIENTO POR LOTES
3: **for** $i <= \mu$ **do** ▷ Bucle hasta llegar al máximo de iteraciones
▷ Calcular el primer y último cuadro de un grafo multipartito para los procesos pares e impares
4: **if** $p_{id}\,\%\,2$ **then**
5: $f_a \leftarrow k * p_{id} + i + f_a$
6: $f_o \leftarrow f_a + k - 1$
7: **else**
8: $f_a \leftarrow k * p_{id} + k * p * i + f_a$
9: $f_o \leftarrow f_a + k - 1$
10: **end if**
11: **procedure** SEGMENTACIÓN EN PARALELO
12: **if** $p_{id}\,\%\,2$ **then**
13: **for** $j < k$ **do**
14: $f_n \leftarrow i + f_a$ ▷ Número de cuadro actual
15: $C \leftarrow \sigma(f_n)$ ▷ Aplicar función de segmentación al cuadro actual para encontrar los contornos
16: **for** $j < n$ **do** ▷ Barrer los contornos para generar el modelo de los nodos
17: Generar modeloN_j
18: $g_i[j] \leftarrow N_j$
19: **end for**
20: **end for**
21: Enviar grafo multipartito G desde proceso
22: p_i a proceso p_{id+1}
23: **end if**
24: **end procedure**

y 20 se realiza la segmentación y creación de los modelos de los objetos para el grafo multipartito correspondiente al proceso par y en la linea 21 se realiza el envío del grafo multipartito al proceso impar adyacente. En la sección correspondiente al rastreo, el grafo multipartito se recibe en la linea 27, de las lineas 31 a 70 se indica la comunicación entre procesos impares para propagar las identidades de los objetos detectados como jugadores donde se aplica al inicio una función de correspondencia bipartita para asignar las identidades recibidas del proceso impar previo adyacente. Como se observa en la figura 5.4 el último proceso impar enviá un grafo al primer proceso impar para propagar las identidaes al siguiente lote de grafos multipartitos, esto se realiza en las lineas 32 y 51. La misma función de rastreo de la versión secuencial pero paralelizada con OpenMP se utiliza para propagar las identidades dentro de cada grafo multipartito en las líneas 40, 62 y 67.

```
25:         procedure RASTREO EN PARALELO
26:             if p_id % 2 == 1 then
27:                 Recibir grafo multipatito G desde prceso
28:                 p_{id-1}
29:                 if p ≠ 2 then                          ▷ Caso general de más de 2 procesos
30:                     if f_a ≠ f_f ∨ i == μ then
31:                         if p_id == 1 then
32:                             Recibir grafo G_f
33:                             del proceso p_{id-1}
34:                         else
35:                             Recibir grafo G_f
36:                             del proceso p_{id-2}
37:                         end if
38:                         ς(G_f)      ▷ Función de correspondencia bipartita para propagar identidades
39:                     end if
40:                     τ(G)                                   ▷ Aplicar función de rastreo
41:                 else if i == 1 then                        ▷ Primer iteración
42:                     if p_id < p − 1 then
43:                         Enviar grafo G del cuadro f_o
44:                         del proceso p_id a p_{id+2}
45:                     end if
46:                 else if i ≠ 1 ∨ μ then                 ▷ Casos entre primer y última iteración
47:                     if pid < p − 1 then
48:                         Enviar grafo G del cuadro f_o
49:                         del proceso p_id a p_{id+2}
50:                     else
51:                         Enviar grafo G del cuadro f_o
52:                         del proceso p_id a p_1
53:                     end if
54:                 else                                       ▷ Última iteración
55:                     if pid < p − 1 then
56:                         Enviar grafo G del cuadro f_o
57:                         del proceso p_id a p_{id+2}
58:                     end if
59:                 end if
60:             else                                       ▷ Caso especial para 2 procesos
61:                 if fg ≠ f_a ∪ i ≠ μ then
62:                     ς(fg)                          ▷ Aplicar función para propagar identidades
63:                     τ(G)                                   ▷ Aplicar función de rastreo
64:                     G_l ← G(f_o)
65:                 else
66:                     if i = μ − 1 then
67:                         ς(G_f)                     ▷ Aplicar función para propagar identidades
68:                         τ(G)                               ▷ Aplicar función de rastreo
69:                     end if
70:                     G_l ← G(f_o)
71:                 end if
72:             end if
73:         end procedure
74:     end for
75: end procedure
```

5.3. Validador

El validador es el programa utilizado para comparar los datos anotados manualmente, considerados como las referencias válidas de las posiciones de jugadores, con las posiciones generadas a partir de los algoritmos de rastreo. Según se observa en la literatura profesional, no existe una métrica estándar para la validación de la precisión de los algoritmos de rastreo de fútbol, por lo tanto a modo de tener una comparación con la gran mayoría de los resultados públicados, la herramienta de validación calcula las métricas más utilizadas a partir de un conjunto de datos de referencia (*ground-truth*) y el conjunto de anotaciones generadas por un algoritmo. Este programa recibe como parámetros de entrada el archivo de anotaciones, el archivo de trayectorias generado por el algoritmo y un archivo de configuración; la salida es un archivo de reporte con los valores de las métricas de rastreo, todos estos archivos utilizan el formato YAML.

Las métricas que despliega corresponden a:

- FP: False positive
- FN: False negative
- TP: True positive
- TN: True negative
- MS: Misses
- MSM: Missmatches
- Rcll: Recall
- Prcn: Precision
- F1: F1 score
- MOTA: Multiple Object Tracking Accuracy
- MOTP: Multiple Object Tracking Precision

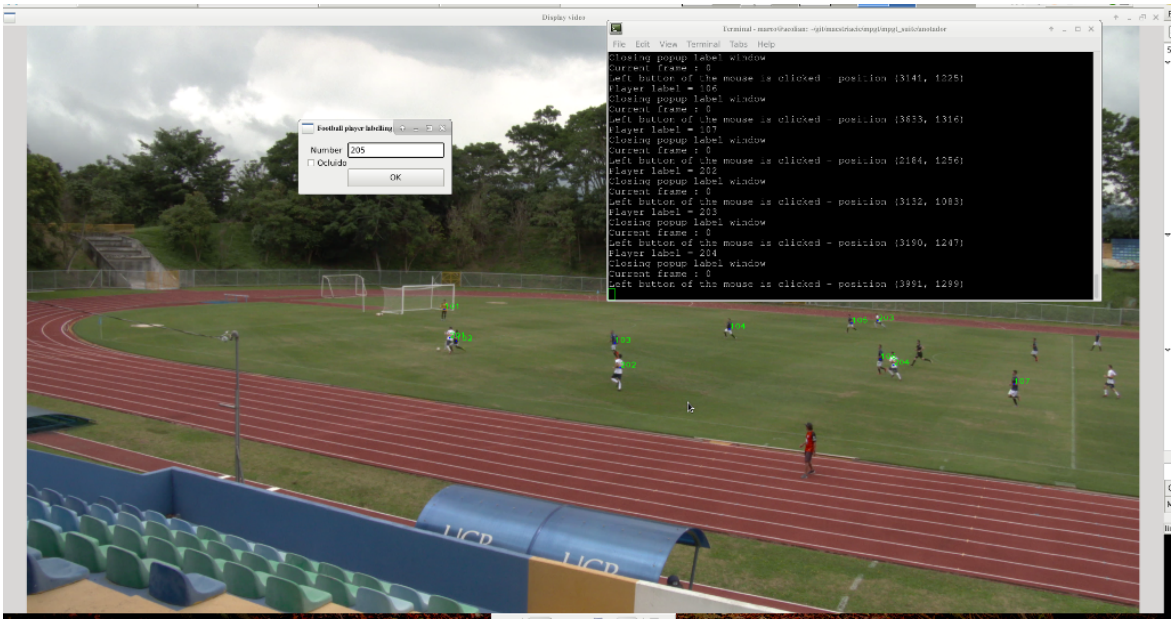


Figura 5.5: Programa de anotación manual de posiciones.

5.4. Anotador

El anotador es un programa de terminal con una interfaz gráfica muy básica que permite realizar anotaciones manuales de las posiciones de los jugadores en un video de fútbol. En la figura 5.5 se observa un cuadro de un video, una ventana para realizar la anotación de la posición previamente marcada con el ratón y la terminal que despliega información de las acciones del proceso de anotación. Este programa recibe como parámetros de entrada un video, el archivo correspondiente a las anotaciones y un archivo de configuración. La salida del programa corresponde a un archivo con las anotaciones de los jugadores, el formato de estos archivos es YAML. Durante la ejecución de este programa están disponibles los siguientes controles o funciones para realizar las anotaciones individuales de las posiciones de jugadores:

- Salto de un cuadro hacia adelante o hacia atrás.
- Salto de un paso de varios cuadros hacia adelante o hacia atrás.
- Incremento o decremento del paso de cuadros por número de cuadros o por tiempo.
- Agregar o borrar anotaciones de jugadores a partir de la posición marcada y de una etiqueta.

5.5. Unificador de Videos

El unificador de videos es el programa para unir las vistas de dos cámaras posicionadas según se explica en este documento para generar las vistas panorámicas de un partido de fútbol, existen alternativas comerciales como Panosport (Over, 2019) y Once Stitcher (Football, 2019), sin embargo tienen un costo alto que los hacen inaccesibles. Recibe como parámetros de entrada los videos de las cámaras y un archivo de configuración, la salida corresponde al video panorámico. El algoritmo para combinar las vistas de las cámaras se basa en una transformación geométrica que encuentra una función que adapta los puntos de un trapezoide a puntos de un rectángulo de una forma continua con los siguientes pasos:

1. Se consideran las esquinas inferiores del trapezoide, estas determinan un ángulo de rotación respecto al eje x, que se utiliza para rotar la imagen de tal forma que el lado correspondiente a las esquinas inferiores sea paralelo al eje x.

2. Usando las esquinas inferiores y la esquina superior de la linea media de campo, se determina el ángulo que forman estos tres puntos, con este ángulo se encuentra una matriz de shear, forzando los lados formados por estos 3 puntos para que sean rectos.

3. Usando los mismos puntos mencionados anteriormente pero con las nuevas coordenadas, se vuelve a transformar las imágenes de tal forma que los puntos correspondientes a los extremos de la linea media coincidan en el eje y.

4. Finalmente se juntan las imágenes y se recorta la imagen resultante para tener solo el campo de juego.

En la figura 5.6 se observa los resultados de las etapas del algoritmo de unificación de videos, a cada una de las imágenes de entrada se les aplica la transformación de rotación para tener a las líneas laterales de forma perpendicular a la línea media del campo y que sean paralelas al eje x, luego se vuelve a realizar una transformación tal que los puntos correspondientes a los extremos de la línea media de campo coincidan, finalmente se unen las vistas y se recorta la vista panorámica para encajar las dimensiones de la misma en las dimensiones permitidas por el codec de video. Cabe destacar que la selección de los puntos es de forma manual utilizando la misma aplicación, primero se seleccionan los puntos correspondientes a las esquinas del campo y los extremos de la línea media en cada vista, luego se seleccionan los puntos que recortan la vista panorámica.

(a) Imágen de entrada izquierda

(b) Imágen de entrada derecha

(c) Imágen izquierda transformada

(d) Imágen derecha transformada

(e) Imágenes transformadas unificadas

(f) Imágenes unificadas recortadas

Figura 5.6: Proceso de unificación de videos, a las imágenes de entrada se les aplican matrices de transformación y luego se juntan para generar una vista panorámica recortada.

(a) Vista Izquierda **(b)** Vista central **(c)** Vista derecha

Figura 5.7: Vistas laterales de la base de datos ISSIA

5.6. Traductor ISSIA

ISSIA-C.N.R. es un conjunto de videos con anotaciones asociadas que se encuentra disponible para uso no comercial. Los videos son las grabaciones de las perspectivas individuales de seis cámaras, estos videos consisten de 3000 cuadros capturados a 25 FPS, las anotaciones son manuales de 22 jugadores y 3 árbitros. En la figura 5.7 se muestran las vistas de tres cámaras correspondientes a un lado, estas vistas tienen el problema que no cubren todo el campo de juego al omitir regiones entre las vistas de las cámaras. El traductor ISSIA es el componente encargado de realizar la traducción del formato de salida basado en XML del conjunto de datos ISSIA-C.N.R. en el formato basado en YAML utilizado en varios de los componentes de la plataforma de rastreo de jugadores de fútbol.

Capítulo 6

Resultados y discusión

6.1. Conjunto de datos

En las tablas 4.3 y 4.4 se indican los partidos disponibles, no todos los partidos son aptos por distintas razones que se discuten a continuación. Como se menciona en el capítulo 4, aquellos provenientes de transmisiones de televisión (tabla 4.4) no se adaptan a las condiciones contempladas en el algoritmo de rastreo al no capturar todo el campo de juego, por lo tanto no se consideran como fuentes de datos para el análisis de resultados.

En las figuras 6.1 y 6.2 se muestran los partidos grabados en el estadio nacional de Costa Rica; en las figuras 6.3, 6.4 y 6.5 se muestran aquellos grabados en el estadio ecológico de la U.C.R., estos si abarcan las condiciones iniciales necesarios para utilizar el algoritmo de rastreo. En estas figuras están tanto las vistas individuales de cada cámara como también la vista panorámica obtenida al utilizar el unificador de videos. Como se indica en la descripción del unificador de videos, se realizan transformaciones geométricas de perspectiva para lograr obtener una vista panorámica, estas transformaciones no contemplan los efectos por diferencias en la profundidad de las tomas, por lo tanto al tener diferentes valores de acercamiento configurados en las cámaras se produce una deformación que no es posible corregir con este algoritmo como se muestra en la figura **??**, en el caso de que este tipo de deformaciones no sean muy profundas como en la figura **??** se puede considerar como aceptable, debido a que la etapa de segmentación del algoritmo de rastreo se detectan como un solo objeto.

En el primer partido grabado en el estadio ecológico de la U.C.R., correspondiente al amistoso entre los equipos masculinos de la U.C.R. y UACA, se utilizó una configuración de cámaras distinta a la propuesta en la metodología, en este caso las vistas de las cámaras son cruzadas, a esto se refiere a que la cámara ubicada al lado derecho graba la mitad izquierda del campo de juego mientras la cámara ubicada al lado izquierdo graba la mitad derecha del campo de juego. La combinación de esta

configuración con el plano donde se ubican las cámaras en este estadio y diferentes valores de acercamientos producen otro tipo de deformación como las que se muestran en la figura 6.6. En esta figura se observa como las proporciones de los jugadores son considerablemente diferentes, por lo tanto este partido queda descartado como fuente de datos.

En la figura 6.7 se muestra el resultado de juntar las vistas laterales de la base de datos ISSIA por medio del programa ffmpeg obteniendo un video con 5744x1080 pixeles de dimensión, al inspeccionar las anotaciones manuales correspondientes a este video se observa un error significativo en el posicionamiento de algunas anotaciones con respecto a las posiciones reales de los jugadores en dicho video, como se observa en la figura **??**, esto debe ser considerado al calcular las métricas de precisión. Un problema del modo panorámico del conjunto de datos ISSIA es la duplicación de las identidades por la superposición parcial entre cámaras como se observa en la figura **??** con el jugador etiquetado con la identidad 107.

6.2. Robustez

En esta sección se hace una evaluación de la precisión que tiene el algoritmo en rastrear correctamente los objetos de interés.

En la tabla 6.1 se resumen los resultados obtenidos por otras investigaciones en términos de la robustez de los algoritmos, como se menciona previamente, no existe un acuerdo sobre el conjunto de datos y las métricas para tener una comparación real del desempeño de los algoritmos, muchos de los conjuntos de datos no están disponibles. La única base de datos pública disponible para uso no comercial es ISSIA y es utilizada para validar los resultados de varios algoritmos de rastreo. En (Baysal & Duygulu, 2016) se indica que las métricas de error son aproximadas y utilizan tres cámaras para tener la vista panorámica, obtienen 18 % para FP y 12 % para FN. En (Shitrit et al. 2014) obtienen 18 % para FP y 18 % para FN mientras que en (Berclaz, Fleuret, Turetken & Fua, 2011) los resultados son 18.5 % para FP y 18 % para FN. Como se indica previamente, este conjunto de datos presenta el problema de la superposición de algunas secciones y la ausencia de otras secciones del campo en las vistas de cámaras adyacentes de un mismo lado, además existe un desplazamiento de las anotaciones de algunos jugadores, por lo tanto para el algoritmo de rastreo de jugadores con grafos multipartitos se permite un error del 4 % para las posiciones de jugadores obteniendo un

Tabla 6.1: Algoritmos de rastreo y métricas de precisión

Autor	Algoritmo	Datos	Valor y Métrica
Manafifard, H.Ebadi y Moghaddam, 2017	MHT PSO	Spagnolo dataset	80 % MOTA
Hoak, Medeiros y Povinelli, 2017	MBFILH	AFL	66.3 % MOTA
Elferink, 2017	Multi Camera Fusion	15 segs Heracles-Twente	97.38 % MOTA
Baysal y Duygulu, 2016	FPF	ISSIA	18 %FP 12 %FN
Musa, Salleh, Bakar y Watada, 2016	MBPF	-	19.94 % FN 0.17 FP
Bozorgtabar y Goecke, 2016	Discoer dense	AFL	66.8 % MOTA
Sabirin, Sankoh y Naito, 2015	Attribute Matching	7 x 170 frames scenes 4K-HD	0.834 Precision
Manafifard, Ebadi y Moghaddam, 2015	Discrete Particle Swan	6 sequences (491 frames)	0.942 Precision
Manafifard, Ebadi y Moghaddam, 2015	Graph optimization	1900 frames	79.43 % MOTA
Shitrit, Berclaz y Fleuret, 2014	T-MCNF	ISSIA	18 % FP 18 % FN
Milan, Gade, Dick, Moeslund y Reid, 2014	Global Multi Tracker	AFL	41.4 % MOTA
Hermann, Hoernig y Radig, 2014	Local maxima on confidence map	ISSIA	75 % MOTA
Hoseinnezhad, Vo, Vo y Suter, 2012	Multi-Bernoulli filtering	2500 PETS	4 % FNR
Berclaz, Fleuret, Turetken y Fua, 2011	KSP	ISSIA	18.5 % FP 18 %FN

19.70 % para FP y 17.49 % para FN, estos resultados se extraen al procesar 2591 cuadros con 66476 jugadores anotados. En la figura 6.8 se observa un cuadro con jugadores rastreados, los objetos con identidades 14 y 18 corresponden a secciones de las líneas de campo no filtradas en la etapa de segmentación del algoritmo, en gran medida los falsos positivos corresponden a este tipo de objetos mientras tanto el valor correspondiente a los falsos negativos se debe a jugadores no detectados o eventos de oclusión donde no se identifica a uno o varios jugadores, como por ejemplo al objeto con la identidad 5.

En la figura 6.9 se muestra una secuencia de imágenes correspondientes al rastreo de varios jugadores de un video por un periodo de 6 segundos para una escena de transición, esta secuencia corresponde a cada 10 cuadros durante este periodo. En esta secuencia se observan algunos secciones de las líneas de campo que no fueron filtradas en la etapa de segmentación, se consideran detecciones falsas. Como se observa en la misma secuencia el seguimiento de un jugador sin oclusiones es una tarea trivial y que el rastreador lo realiza de forma correcta, incluso en ocasiones cuando no es detectado temporalmente en la etapa de segmentación, al utilizar la matriz de coincidencias o con la detección de este tipo de evento con el barrido del grafo multipartito para recuperar la identidad, un ejemplo de este rastreo individual se observa con los porteros cuyas identidades son 19 y 22. El algoritmo de rastreo resuelve también el caso de oclusiones entre jugadores de distintos equipos como por ejemplo, los jugadores cuyas identidades son 6 y 9 ó 12 y 13 en la secuencia comprendidos entre la figura 6.9a y la figura 6.9d. El algoritmo puede resolver de forma parcial las oclusiones entre tres o más objetos como se observa entre la figura 6.9d y 6.9h entre los jugadores con identidades 14,15 y 23, en este caso las identidades de los jugadores del mismo equipo se intercambian.

6.3. Aceleración, velocidad y eficiencia

El objetivo principal de un algoritmo de rastreo de jugadores de fútbol es la identificación precisa y el rastreo de cada objeto en el campo de juego, sin embargo, la complejidad de los algoritmos así como también los datos de entrada pueden hacer de estos algoritmos poco prácticos, al obtener los resultados en un tiempo considerablemente largo.

En la tabla 6.2 se tienen todos aquellos algoritmos que reportan tiempos de procesamiento y razones de cuadros por segundo encontrados en la literatura profesional. Se observa que Matlab es el lenguaje de implementación más común, Matlab es un ambiente y lenguaje de programación que permite realizar prototipado de algoritmos al tener una gran cantidad de bibliotecas que aceleran el proceso de programación, validación funcional y depurado. Sin embargo al ser un lenguaje interpretado que se va compilando en tiempo de ejecución agrega un gasto computacional que incide como un bajo desempeño, esto se observa en esta tabla donde los algoritmos con implementaciones en esta plataformas no llegan a superar los 4 FPS. Solo dos algoritmos se encuentran implementados en el lenguaje de programación C/C++, una de estas implementaciones alcanza el valor de 14 FPS, aun por debajo del valor deseado de 30 FPS. Todas estas implementaciones utilizan conjuntos de datos cuyas resoluciones y calidades están por debajo a los datos de entrada del presente trabajo.

(a) Imágen de entrada izquierda CRC/JMC

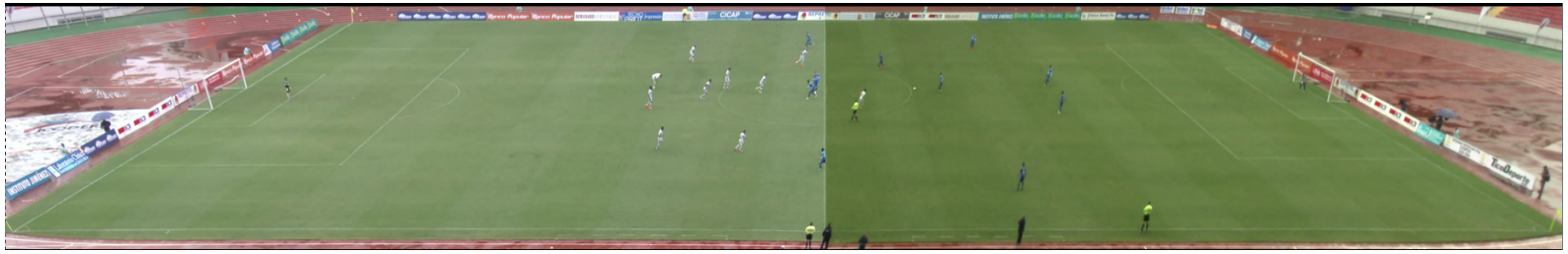
(b) Imágen de entrada derecha CRC/JMC

(c) Imágenes unificadas recortadas CRC/JMC

Figura 6.1: Vistas de entrada y vista panorámica resultante para el partido Costa Rica - Jamaica.

(a) Imágen de entrada izquierda UCR/HRD

(b) Imágen de entrada derecha UCR/HRD



(c) Imágenes unificadas recortadas UCR/HRD

Figura 6.2: Vistas de entrada y vista panorámica resultante para el partido UCR - Herediano.

(a) Imágen de entrada izquierda UCR-UACA 1

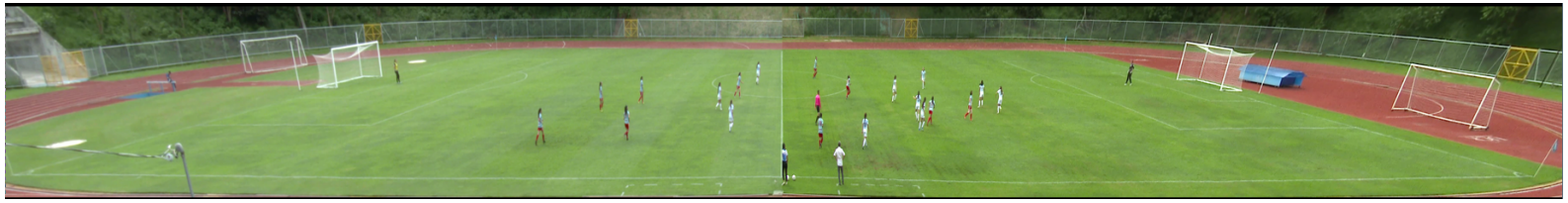
(b) Imágen de entrada derecha UCR-UACA 1

(f) Imágenes unificadas recortadas UCR-UACA 1

Figura 6.3: Vistas de entrada y vista panorámica resultante para el primer partido entre UCR - UACA 1.

(a) Imágen de entrada izquierda UCR-Fem.

(b) Imágen de entrada derecha UCR-Fem.



(c) Imágenes unificadas recortadas UCR-Fem.

Figura 6.4: Vistas de entrada y vista panorámica resultante para el partido UCR Femenino.

(a) Imágen de entrada izquierda UCR-UACA 2

(b) Imágen de entrada derecha UCR-UACA 2

(f) Imágenes unificadas recortadas UCR-UACA 2

Figura 6.5: Vistas de entrada y vista panorámica resultante para el primer partido entre UCR - UACA 2.

(a) Deformación 4 **(b)** Deformación 5

Figura 6.6: Deformaciones por tomas cruzadas de cámaras y por un plano muy bajo.

Figura 6.7: Vistas laterales de la base de datos ISSIA concatenadas.

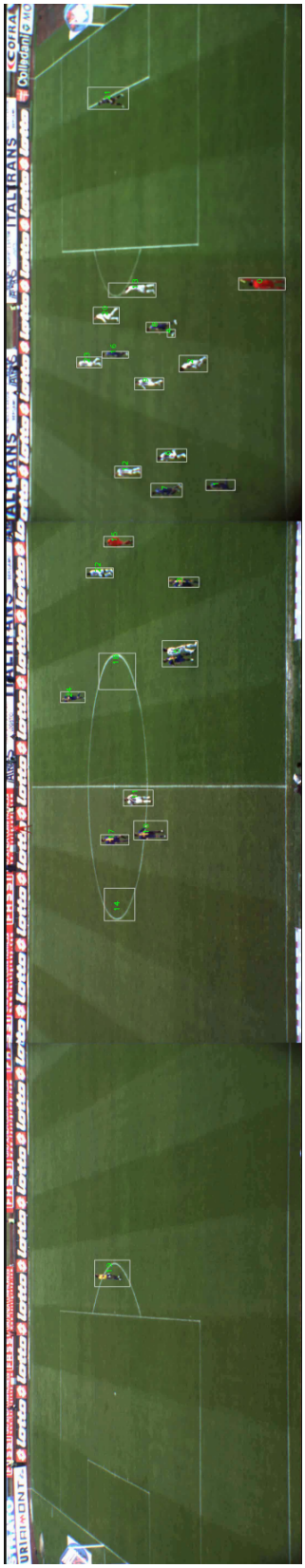
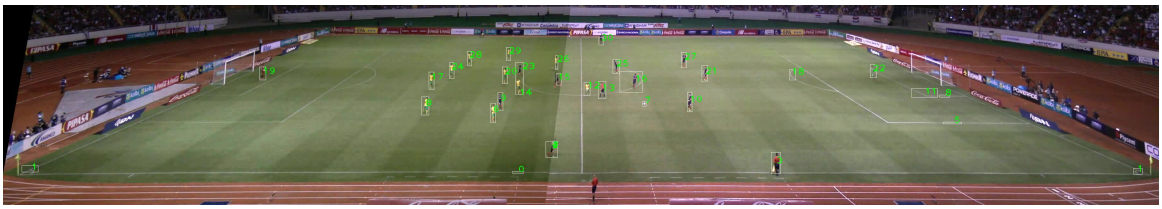
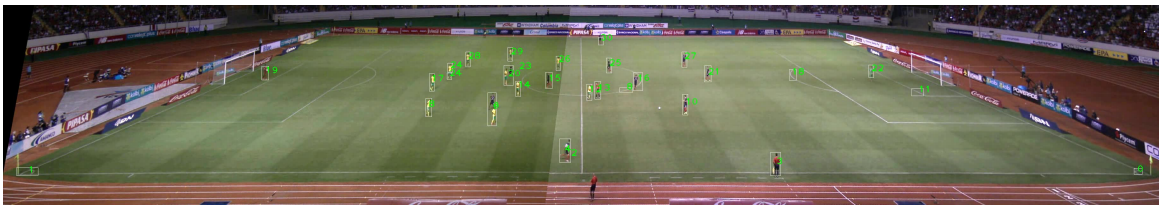


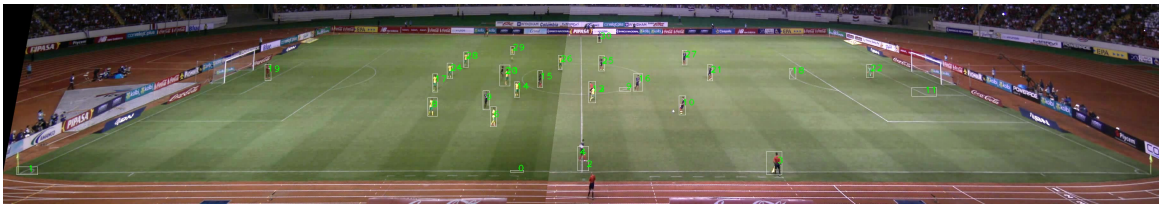
Figura 6.8: Vista panorámica ISSIA y jugadores rastreados con el algoritmo de grafos multipartitos.



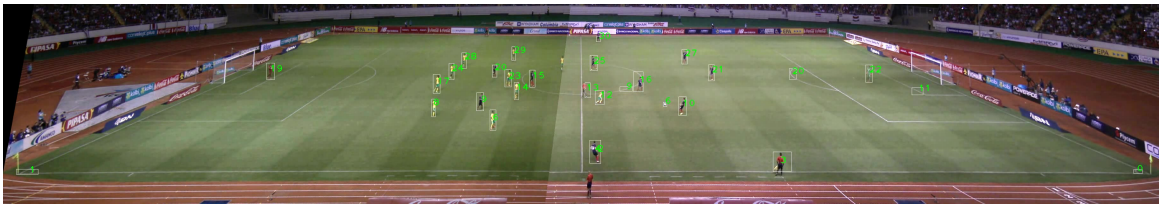
(a) Cuadro i+0



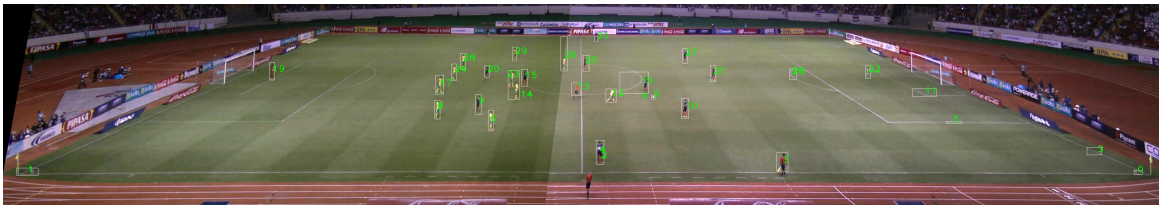
(b) Cuadro i+10



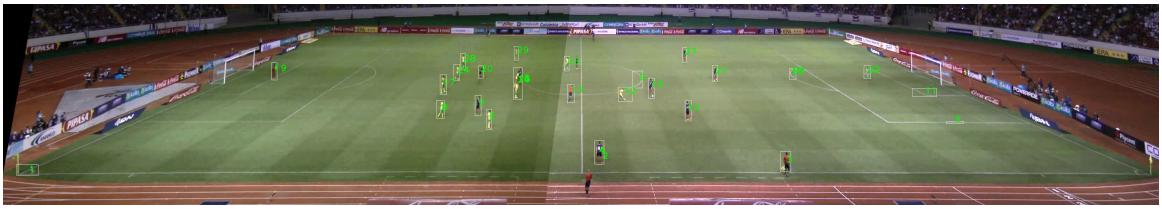
(c) Cuadro i+20



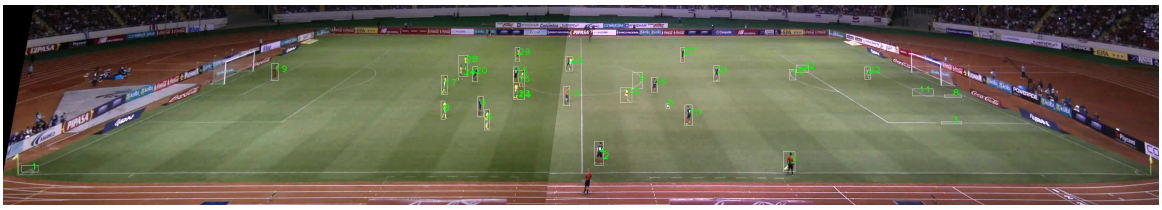
(d) Cuadro i+30



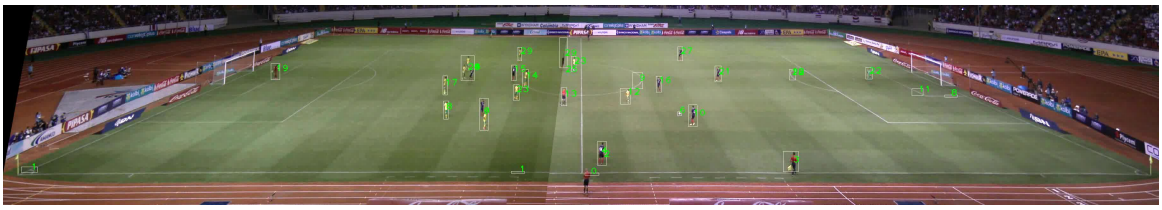
(e) Cuadro i+40



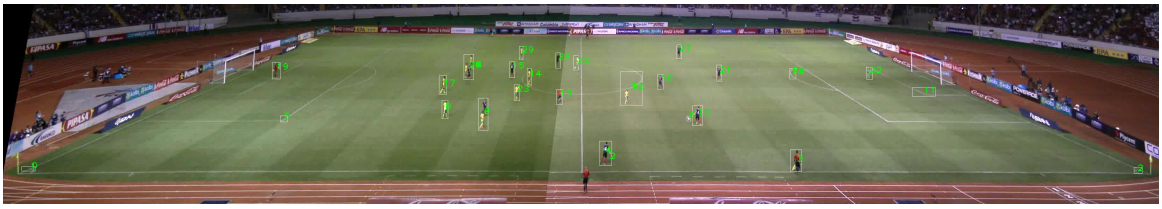
(f) Cuadro i+50



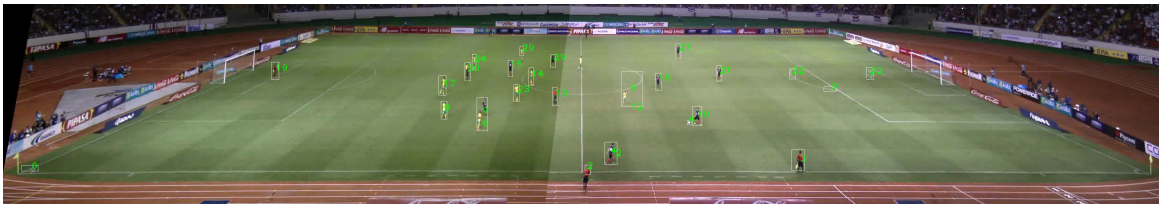
(g) Cuadro i+60



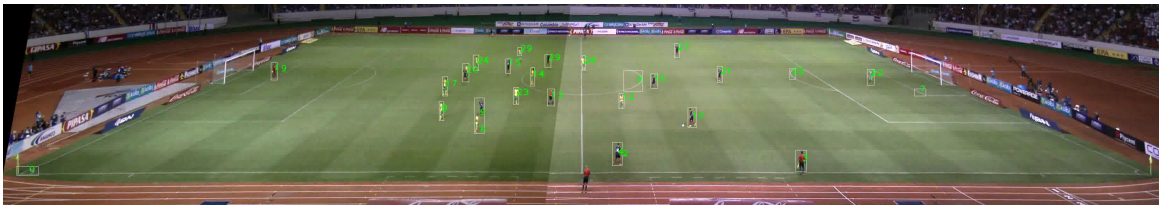
(h) Cuadro i+70



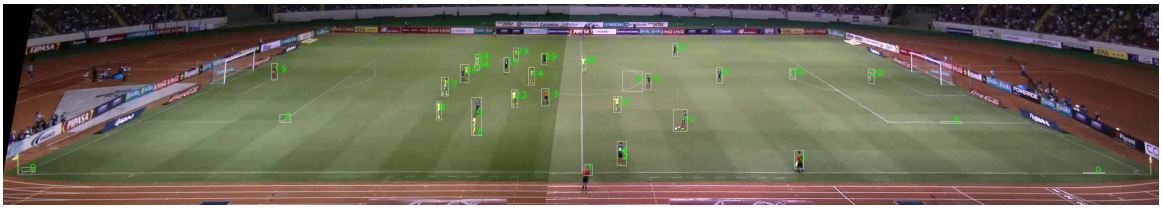
(i) Cuadro i+80



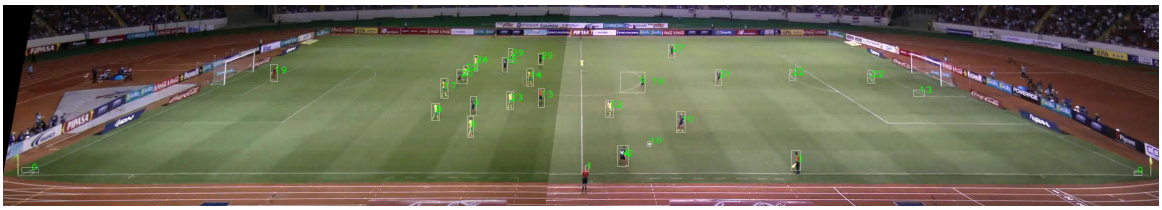
(j) Cuadro i+90



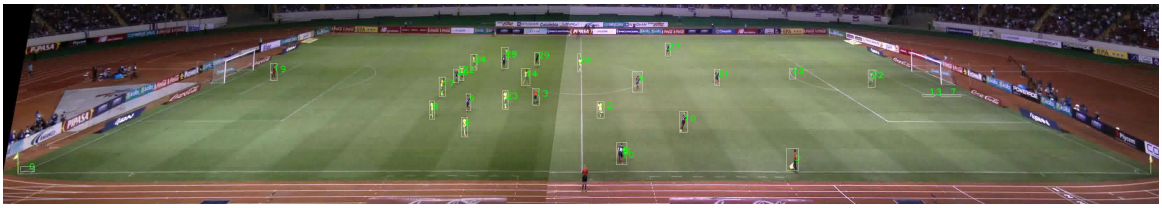
(k) Cuadro i+100



(l) Cuadro i+110



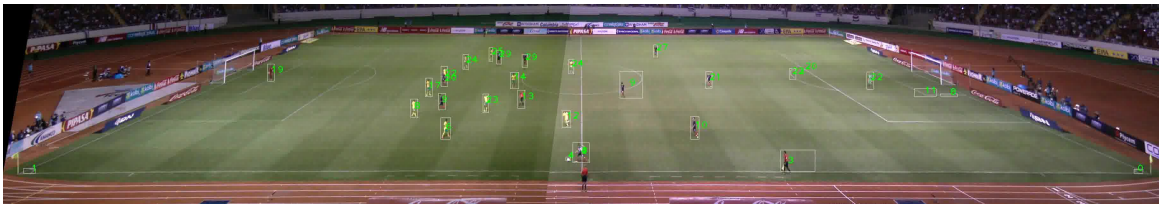
(m) Cuadro i+120



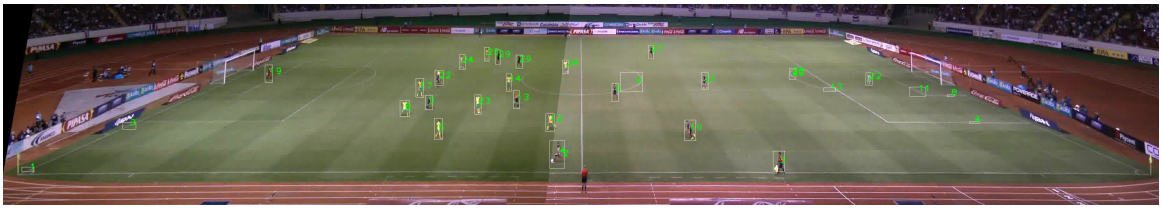
(n) Cuadro i+130

(ñ) Cuadro i+140

(o) Cuadro i+150



(p) Cuadro i+160



(q) Cuadro i+170

Figura 6.9: Secuencia de rastreo de jugadores de fútbol correspondiente a seis segundos de una escena de transición, saltando cada diez cuadros en un video de treinta cuadros por segundo.

Tabla 6.2: Velocidad de algoritmos de rastreo

Autor	Algoritmo	Lenguaje	Datos	Hardware	FPS
Lefèvre, Fluck, Maillard y Vincent, 2000	Fast snake	Matlab	100 imágenes de 384x288 px	Intel Celeron 600, 128 MB RAM	0.016
de Vos y Brink, 2009	Motion detector and hierarchical particle filter	Matlab	400 imágenes	-	4
Yuan, Emmanuel y Fang, 2019	Backward model validation	Matlab	PETS2009 dataset	Intel E5-1650 3.2GHz, 16 GB RAM	3
Martín y Martínez, 2014	Semi supervised system	Matlab	ISSIA dataset	Intel Core I7 2.66GHz, 6 GB RAM	3.3
Najafzadeh, Fotouhi y Kasaci, 2015	Adaptive Kalman Filter	Matlab	Azadi dataset 720x576	Intel Core 2 DUO 2.2GHz, 4 GB RAM	-
Ma, Feng y Wang, 2018	Fully-convolutional siamese network	Matlab	FT dataset	Intel Core I7-4720HQ	4
Baysal y Duygulu, 2016	Model Field Particles	C++	-	Intel Core I7 2.6GHz	14
Shitrit, Berclaz y Fleuret, 2014	Multi-commodity network flow	C++	ISSIA dataset	3 GHz	3.95

Tabla 6.3: Tiempos de ejecución para el algoritmo secuencial

Resolución	Tiempo de ejecución (s)	Tiempo de segmentación (s)	Tiempo de rastreo (s)
4K	2267	2229	38
FHD	542	530	12
HD	286	283	3

Tabla 6.4: Resultados de rastreo en paralelo para un video 4K

Hilos	Tiempo de ejecución (s)	Aceleración	Eficiencia	FPS	Tiempo de rastreo (s)
128	117.85	19.24	0.15	21.71	2.07
64	176.69	12.83	0.20	14.48	2.96
32	218.51	14.45	0.45	12.44	3.89
16	426.27	7.41	0.46	6.38	5.20
8	519.44	6.23	0.78	5.39	8.05
4	1215.93	2.69	0.67	2.33	16.91
2	2190.06	1.49	0.75	1.30	37.69
1	2267.00	1.00	1.00	1.27	38.00

Tabla 6.5: Efecto del tamaño del grafo multipartito

Tamaño	Hilos	Tiempo de ejecución (s)	Aceleración	Eficiencia	FPS	Tiempo de rastreo (s)
20	128	117.85	19.24	0.15	21.71	2.07
10	128	164.27	13.80	0.11	15.58	14.19
5	128	217.96	10.40	0.08	11.74	15.19

Por lo mencionado anteriormente, se inspecciona el efecto de paralelizar el algoritmo de rastreo bajo distintas condiciones como la cantidad de hilos, resolución de los datos de entrada y el tamaño de los grafos multipartitos sobre el tiempo de ejecución, la aceleración y eficiencia.

En la tabla 6.3 se observan los tiempos de ejecución del algoritmo de rastreo de jugadores de fútbol en su versión secuencial para tres resoluciones de video distintas y los tiempos que duran los dos principales procedimientos del algoritmo. En la versión secuencial del algoritmo no más del 2 % del tiempo de ejecución se invierte en el rastreo. De las tablas 6.3, 6.4 y 6.5 se evidencia como la sección de segmentación del algoritmo es el cuello de botella al realizar funciones con un alto costo computacional. En particular, la detección de contornos ocupa un 36.66 % del tiempo de ejecución de esta sección, este dato se obtuvo al realizar un perfilado de la aplicación correspondiente a esta versión del algoritmo.

De la tabla 6.4 se tiene que la paralelización de datos al ser distribuidos de forma equitativa entre los procesos pares y la paralelización de tareas de los procedimientos de segmentación y rastreo,

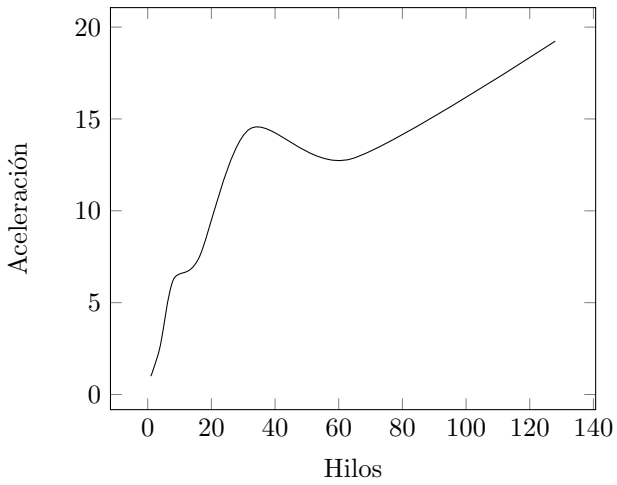


Figura 6.10: Aceleración para video 4K.

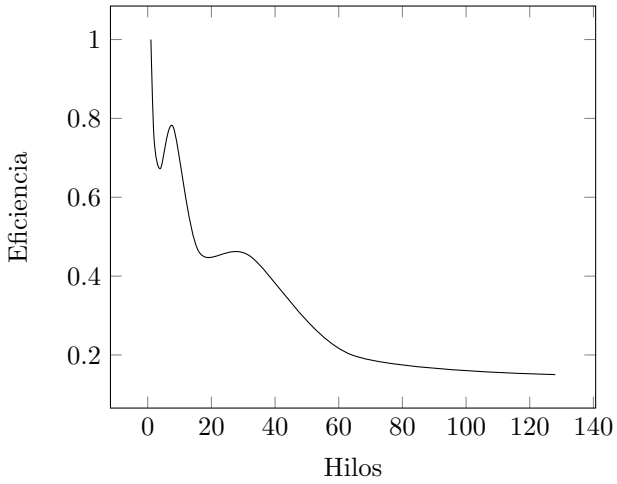


Figura 6.11: Eficiencia para video 4K.

mejoran la aceleración del algoritmo y la razón de cuadros por segundos que puede procesar. Si la relación entre los hilos pares e impares se mantiene, el tiempo de prcesamiento se encuentra en el rango del 1.22 % al 1.78 % del tiempo total de ejecución, esto se debe a que la minimización del grafo multipartito utiliza funciones que son computacionalmente simples en comparación con las funciones de segmentación, además la propagación de identidades no consume muchos recursos a pesar de la serialización de las estructuras de datos que representan a los nodos, esto es necesario en cualquier tipo de comunicación de MPI.

El crecimiento en la aceleración basado en el tamaño del grafo multipartito se debe a una menor

Tabla 6.6: Resultados de rastreo en paralelo para resoluciones de video

Resolución	Hilos	Tiempo de ejecución (s)	Aceleración	Eficiencia	FPS	Tiempo de rastreo (s)
4K	128	117.85	19.24	0.15	21.71	2.07
FHD	128	31.10	24.15	0.19	82.28	2.00
HD	128	17.71	23.99	0.19	144.47	1.95

cantidad de comunicaciones entre los procesos pares e impares y a las comunicaciones de los procesos impares adyacentes, esto implica a la vez un uso mayor de la memoria como se muestra en la tabla 6.5, se debe tener cuidado con el uso de la memoria al ser un recurso limitado por lo que se restringe las pruebas a un tamaño de 20. La información de la tabla 6.4, figura 6.11 y de la figura 6.10 muestran como existen una aceleración consistente, pero con una penalización en la eficiencia debido a las secciones secuenciales del algoritmo.

Como se observa de la figura 6.10 el algoritmo en paralelo tiene una tendencia positiva a escalar a un número mayor de procesos, sin embargo, se debe tener precaución con el tamaño del grafo multipartito en términos del consumo de memoria, a pesar de mostrar mejores resultados al incrementar su tamaño de acuerdo a la tabla 6.5. La aceleración se percibe como lineal dentro de un nodo pero cuando se agregan más nodos, la aceleración se afecta por las comunicaciones entre los procesos que reciden en nodos diferentes y tienen que compartir información.

De la tabla 6.6 se observan los resultados de la ejecución con distintas resoluciones, es de esperar que al aumentar la resolución aumente el tiempo que dura el algoritmo en procesar la información para una misma cantidad de hilos y se disminuya a la vez la frecuencia de cuadros por segundo. A pesar de lo anterior, el algoritmo logra obtener 21.71 FPS para un video 4K, siendo un valor superior a otros algoritmos de rastreo para una resolución más alta de datos.

Capítulo 7

Conclusiones, recomendaciones y trabajo futuro

7.1. Conclusiones

En este trabajo se presenta una investigación profunda de los algoritmos de rastreo de jugadores de fútbol y se desarrolla un algoritmo. En este capítulo se resumen las conclusiones, contribuciones y finalmente se realizan recomendaciones sobre el trabajo futuro.

Se propone un algoritmo automatizado de rastreo y clasificación de jugadores utilizando grafos multipartitos y videos de ultra alta definición, este algoritmo utiliza las características de la forma, color, posición y dirección para construir un vector de descriptores para cada objeto detectado como jugador de fútbol, a partir de los vectores se crea la abstracción del grafo multipatito para realizar el rastreo de los jugadores.

Se desarrolla una plataforma compuesta por varios elementos que complementan al algoritmo de rastreo de jugadores. Estos elementos como la herramienta de anotación manual y de validación no solo tienen utilidad para los algoritmos de rastreo de jugadores de fútbol al incluir características generales para el rastreo de múltiples objetos como las métricas, por lo anterior no es necesario realizar modificación avanzadas a estas herramientas para ser utilizadas en otro tipo de aplicaciones como los algoritmos de rastreo de células.

De los resultados obtenidos con la base de datos ISSIA se observa que los valores de las métricas son similares a otros algoritmos de rastreo, que utilizan esta misma base de datos y se puede inferir que obtiene estos resultados en un menor tiempo, por ejemplo (Shitrit et al. 2014) analiza este conjunto de datos de 6203520 pixeles con una razón de 3.95 FPS, mientras la versión en paralelo del algoritmo

de rastreo con grafos multipartitos procesa 8294400 pixeles a 21.71 FPS.

En la literatura profesional de algoritmos de rastreo de jugadores de fútbol no existen referencias de técnicas de paralelización para sistemas de memoria distribuida. A pesar de que la complejidad del algoritmo 2 (versión en paralelo) es mucho mayor que el algoritmo 1 (versión secuencial), los beneficios asociados a esta técnica de aceleración al exhibir tiempos de ejecución mayores a algoritmos del estado del arte llevan a la conclusión de la necesidad de considerar los sistemas híbridos de paralelización, en especial para las condiciones dadas y el tipo de datos de entrada.

7.2. Recomendaciones

Este trabajo no resuelve por completo la problemática más compleja del rastreo de jugadores de fútbol: las oclusiones. Es evidente del estudio realizado en el estado de la cuestión que sigue siendo el mayor reto de este campo al mantenerse como un problema abierto para el caso de oclusiones con más de dos objetos. El algoritmo de rastreo con grafos multipartitos utiliza el registro de los posibles identificadores de jugadores durante una oclusión en conjunto con un comparador de plantillas de los jugadores ocluidos como parte de la resolución de este tipo de evento, el comparador de plantillas utiliza el método basado en el cuadrado de la diferencia de intensidades, se sugiere utilizar el método propuesto en (Oron, Dekel, Xue, Freeman & Avidan, 2017). Es apropiado profundizar en la resolución de estos eventos en futuros trabajos relacionados.

La implementación de las etapas de segmentación y rastreo utilizan la versión 2.4 de la biblioteca OpenCV, en el corto plazo debería realizarse una versión actualizada de la implementación utilizando la versión 3.4 o 4.1 para tener acceso a diferentes versiones de algoritmos de visión por computadora y nuevas características. Los algoritmos de Yolo (Redmon & Farhadi, 2018) y de Mask R-CNN (He, Gkioxari, Dollár & Girshick, 2017) se pueden implementar o tener acceso con una versión más actual, se consideran como métodos alternativos de segmentación y detección válidos a explorar.

Es recomendable mejorar la experiencia de usuario para las aplicaciones de anotación y validación, considerando aquellos usuarios que no disponen de conocimientos técnicos en el uso de la terminal, por lo tanto a estas herramientas se les debe agregar una interfaz gráfica intuitiva sustituyendo la existente basada en GTK2 por una en QT y una documentación que explique como usarlas.

Para la versión en paralelo del algoritmo se recomienda investigar otros esquemas de distribución de tareas y datos que mejoren la aceleración debido al alto impacto que tiene la segmentación en el algoritmo en paralelo, por ejemplo al realizar un balance de carga disponiendo de más hilos para el procedimiento de segmentado, además se recomienda implementar esta sección utilizando programación avanzada de GPGPU.

Con respecto al conjunto de datos, se debe utilizar una perspectiva de las cámaras tal que reduzca la rotación de las imágenes para ser unidas en una vista panorámica, por ejemplo evitando las perspec-

tivas diagonales o tomas cruzadas. Además se aconseja utilizar un plano con una mayor elevación para reducir las oclusiones parciales o totales entre jugadoras y de esta forma a la vez tener una perceptiva más similar a las capturas realizadas en las transmisiones de televisión. Para facilitar la sincronización de las vistas se debe utilizar la función *TC Link* de las cámaras PXW-Z100, de otra forma se debe utilizar la función de sincronización manual del unificador de videos.

No se recomienda el uso de la base de datos ISSIA debido al problema de las anotaciones manuales que contienen errores que no reflejan las posiciones reales con precisión, además no captura toda la interacción en el campo perdiendo información importante para el rastreador. El único beneficio asociado con estos datos es la posibilidad de comparar los resultados con otros algoritmos de rastreo.

Capítulo 8

Bibliografía

Amdahl, G. (1967). Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. *AFIPS Conference Proceedings*, *30*, 483-485.

Baysal, S. & Duygulu, P. (2016). Sentioscope: A Soccer Player Tracking System Using Model Field Particles. *IEEE Transactions on Circuits and Aystems for Video Technology*, *Vol. 26*(No. 7).

Bebie, T. & Bieri, H. (1998). SoccerMan-Reconstructing Soccer Games from Video Sequences. *ICIP 98. Proceedings. 1998 International Conference on.*

Berclaz, J., Fleuret, F., Turetken, E. & Fua, P. (2011). Multiple object tracking using k-shortest paths optimization. *IEEE Transaction on Pattern Analysis and Machine Intelligence.*

Bernardin, K. & Stiefelhagen, R. (2008). Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics. *EUROSAIP.*

Bishop, C. M. (2006). *Pattern Recognittion and Machine Learning.* Springer Science+Business Media.

Bozorgtabar, B. & Goecke, R. (2016). Efficient multi-target tracking via discovering dense subgraphs. *Computer Vision and Image Understanding*, *144*, 205-216.

Cehovin, L., Leonardis, A. & Kristan, M. (2016). Visual object tracking performance measures revisited. *IEEE Transactions on Image Processing.*

Comaniciu, D. & Meer, P. (2002). Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *24*, 603-619.

Corporation, C. (2017, mayo). TRACAB OPTICAL TRACKING. http://chyronhego.com/sports-data/tracab. Recuperado desde http://chyronhego.com/sports-data/tracab

Czyz, J., Ristic, B. & Macq, B. (2005). A color-based particle filter for joint detection and tracking of multiple objects. *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on.*

de Vos, R. C. H. & Brink, W. (2009). Combining motion detection and hierarchical particle filter tracking in a multi-player sports environment. *Proceedings of the 20th Symposium of the Pattern Recognition Association of South Africa.*

Dearden, A., Demiris, Y. & Grau, O. (2006). Tracking football player movement form a single moving camera using partile fitlers. *Proc. 3rd Eur. Conf. Vis. Media Prod.*

Duda, H. P. & Stork, D. (2001). *Pattern Classification and Scene Analysis* (2nd ed.). Wiley.

Elferink, W. O. (2017, 22 de agosto). *Multi-Camera Tracking of Soccer Players Throug Severe Occlusions* (Tesis de maestría, University of Twente).

Figueroa, P., Leite, N. & Barros, R. M. L. (2004). Tracking soccer players using the graph representation. En *Proceedings of the 17th International Conference on Pattern Recognition* (pp. 787-790).

Figueroa, P., Leite, N. & Barros, R. M. L. (2006). Tracking soccer players aiming their kinematical motion analysis. *Computer Vision and Image Understanding*, *Vol. 101*(Issue 2), 122-135.

Figueroa, P., Leite, N., Barros, R. M. L., Cohen, I. & Medioni, G. (2004). Tracking Soccer Players Using the Graph Representation. En *Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 4 - Volume 04* (pp. 787-790). ICPR '04. Washington, DC, USA: IEEE Computer Society. doi:10.1109/ICPR.2004.950

Football, O. (2019). Once Stitcher. http://www.once.de/products/once-stitcher/2916.

Forum, M. P. I. (2015). *MPI: A Message-Passing Interface Standard*. Message Passing Interface Forum. Recuperado desde https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf

Fu, K.-S. & Rosenfeld, A. (1976). Pattern Recognition and Image Processing. *IEEE Transactions on Computers*, *C-25*, 1336-1337.

Gandón, A., Pons, S. & Ruiz-Shulcloper, J. (2012). *Algoritmos de agrupamiento conceptuales: Un estado del arte*. Cenatav.

Gedikli, S., Bandouch, J., Hoyningen-Huene, N., Kirchlechner, B. & Beetz, M. (2007). An adaptive vision system for tracking soccer players from variable camera settings. *ICVS*.

Gonzalez, R. C. & Woods, R. E. (2008). *Digital image processing*. Pearson Education.

Goszczynska, H. (Ed.). (2011). *Object Tracking*. InTech. doi:10.5772/614

Habibi, Y., Sulistyaningrum, D. R. & Setiyono, B. (2017). A new algorithm for small object tracking based on super-resolution technique. *AIP Conference Proceedings*, *1867*(1), 020024. doi:10.1063/1.4994427. eprint: https://aip.scitation.org/doi/pdf/10.1063/1.4994427

Hare, S., Saffari, A. & Torr, P. H. S. (2011). Struck: Structured Output Tracking with Kernels. En *Proceedings of the 2011 International Conference on Computer Vision* (pp. 263-270). ICCV '11. Washington, DC, USA: IEEE Computer Society. doi:10.1109/ICCV.2011.6126251

He, K., Gkioxari, G., Dollár, P. & Girshick, R. B. (2017). Mask R-CNN. *CoRR*, *abs/1703.06870*. arXiv: 1703.06870. Recuperado desde http://arxiv.org/abs/1703.06870

Hermann, M., Hoernig, M. & Radig, B. (2014). Online Multi-player Tracking in Monocular Soccer Videos. *AASRI Procedia*, *8*, 30-37.

Hess, R. & Fern, A. (2009). Discriminatively Trained Particle Filters for Complex Multi-Object Tracking. En *Proc. IEEE CVPR*.

Hoak, A., Medeiros, H. & Povinelli, R. J. (2017). Image-based multi-target tracking through multi-bernoulli filtering with interactive likelihoods. *Sensors*.

Hoseinnezhad, R., Vo, B.-N., Vo, B.-T. & Suter, D. (2012). Visual Tracking of Numerous Targets via multi-Bernoulli Filtering of Image Data. *Pattern Recogn. 45*(10), 3625-3635. doi:10.1016/j.patcog.2012.04.004

Hu, M.-C., Chang, M.-H., Wu, J.-L. & Chi, L. (2011). Robust Camera Calibration and Player Tracking in Broadcast Basketball Video. *Trans. Multi. 13*(2), 266-279. doi:10.1109/TMM.2010.2100373

Inc., M. A. (2017). Match Analysis - Unrivaled Video and Statistical Analysis for Soccer (Football). http://matchanalysis.com/. Recuperado desde http://matchanalysis.com

Intel Corporation, N. (2004). Threading Methodology: Principles and Practices. *Intel Developer Zone*.

Itoh, H., Takiguchi, T. & Ariki, Y. (2012). 3D Tracking of Soccer Players Using Time-Situation Graph in Monocular Image Sequence. *21st Conference on Pattern Recognition (ICPR 2012)*.

Iwase, S. & Saito, H. (2004). Parallel Tracking of All Soccer Players by Integrating Detected Positions in Multiple View Images. *Proc. 17th ICPR*, 751-754.

Kapanowski, A. & Galuszka, L. (2015, 29 de abril). Weighted graph algorithms with Python. *ArXiv*.

Kristan, M., Perš, J., Perše, M. & Kovačič, S. (2009). Closed-world tracking of multiple interacting targets for indoor-sports applications. *Computer Vision and Image Understanding*, *Vol. 113*(Issue 5), 598-611.

Lefèvre, S., Fluck, C., Maillard, B. & Vincent, N. (2000). A fast snake-based method to track football players. *IAPR International Workshop on Machine Vision Applications*.

Li, Y., Dore, A. & Orwell, J. (2005). Evaluating the performance of systems for tracking football players and ball. *IEEE Conference on Advanced Video and Signal Based Surveillance*.

LLC., S. (2017). STATS Player Tracking. https://www.stats.com/player-tracking/. Recuperado desde https://www.stats.com/player-tracking

Ma, Y., Feng, S. & Wang, Y. (2018). Fully-Convolutional Siamese Networks for Football Player Tracking. *2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS)*.

Mackay, D. (2009). A Library Based Approach to Threading for Performance. *Intel Developer Zone*. Recuperado desde http://software.intel.com/en-us/articles/a-library-based-approach-to-threading-for-performance

Manafifard, M., Ebadi, H. & Moghaddam, H. A. (2017). A survey on player tracking in soccer videos. *Computer Vision and Image Understanding*.

Manafifard, M., Ebadi, H. & Moghaddam, H. A. (2015). Discrete particle swarm optimization for player trajectory extraction in soccer broadcast videos. *Scientia Iranica*.

Manafifard, M., H.Ebadi & Moghaddam, H. A. (2017). Appearance-based hipothesis tracking: Application to soccer broadcast videos analysis. *Signal Processing Image Communication 55*.

Martín, R. & Martínez, J. M. (2014). A Semi-supervised System for Players Detection and Tracking in Multi-camera Soccer Videos. *Multimedia Tools Appl. 73*(3), 1617-1642. doi:10.1007/s11042-013-1659-6

Mattson, T., Sanders, B. & Massingill, B. (2005). *Patterns for Parallel Programming*. Addison-Wesley Professional.

Michael McCool, A. D. R. J. R. (2012). *Structured Parallel Programming - Patterns for Efficient Computation*. Morgan Kaufmann.

Milan, A., Gade, R., Dick, A., Moeslund, T. B. & Reid, I. (2014). Improving Global Multi-Target Tracking with Local Updates. *European Conference on Computer Vision*.

Misu, T., Naemura, M., Zheng, W., Izumi, Y. & Fukui, K. (2002). Robust Tracking of Soccer Players Based on Data Fusion. En *Proceedings of the 16 th International Conference on Pattern Recognition (ICPR'02) Volume 1 - Volume 1* (pp. 10556-). ICPR '02. Washington, DC, USA: IEEE Computer Society. Recuperado desde http://dl.acm.org/citation.cfm?id=839290.842686

Morais, E., Ferreira, A., Cunha, S. A., Barros, R. M., Rocha, A. & Goldenstein, S. (2014). A multiple camera methodology for automatic localization and tracking of futsal players. *Pattern Recognition Letters, vol. 39*, 21-30.

Morais, E., Goldenstein, S., Ferreira, A. & Rocha, A. (2012). Automatic tracking of indoor soccer players using videos from multiple cameras. En *XXV SIBGRAPI Conference on Graphics, Patterns and Images*.

Morimitsu, H., Cesar, R. M. & Block, I. (2015). Attributed Graphs for Tracking Multiple Objects in Structured Sports Videos. *IEEE Iternational Conference on Computer Vision.*

Moya, D. J. P. A. (2012). *Procesamient y Análisis de Imágenes Digitales.* TEC.

Musa, Z., Salleh, M. Z., Bakar, R. A. & Watada, J. (2016). GbLN-PSO and Model-Based Particle Filter Approach for Tracking Human Movements in Large View Cases. *IEEE Transactions on Circuits and Systems for Video Technology.*

Najafzadeh, N., Fotouhi, M. & Kasaci, S. (2015). Multiple Soccer Players Tracking. *2015 International Symposium on Artificial Intelligence and Signal Processing.*

Ok, H.-W., Seo, Y. & Hong, K.-S. (2002). Multiple soccer players tracking by condensation with occlusion alarm probability. En *Proc. Workshop Statist. Methods Vis. Process.*

Open-MPI-Project. (2015). *Open MPI v1.10 documentation.* 1.10.3. Open MPI project. Recuperado desde https://www.open-mpi.org/doc/v1.10

Orazio, T. D., M.Leo, Mosca, N., P.Spagnolo & P.L.Mazzeo. (2009). A Semi-Automatic System for Ground Truth Generation of Soccer Video Sequences. *6th IEEE International Conference on Advanced Video and Signal Surveillance, Genoa, Italy September 2-4 2009.*

Oron, S., Dekel, T., Xue, T., Freeman, W. T. & Avidan, S. (2017). Best-Buddies Similarity—Robust Template Matching Using Mutual Nearest Neighbors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *8*(8), 1799-1813.

Over, S. (2019). Panosport. http://www.panosport.com.

Pacheco, P. S. (2011). *An Introduction to Parallel Programming* (T. Green, Ed.). Morgan Kaufmann.

Pers, J. & Kovacic, S. (2001). Tracking People in Sport: Making Use of Partially Controlled Environment. En *Proceedings of the 9th International Conference on Computer Analysis of Images and Patterns* (pp. 374-382). CAIP '01. London, UK, UK: Springer-Verlag. Recuperado desde http://dl.acm.org/citation.cfm?id=648243.753188

Petsas, P. & Kaimakis, P. (2016). Soccer Player Tracking Using Particle Filters. *IEEE International Symposium on Signal Processing and Information Technology.*

Phillips, C. A. (2015). *Multipartite Graph Algorithms for the Analysis of Heterogeneous Data* (Tesis doctoral, University of Tennesse).

Radakovic, R., Dopsaj, M. & Vulovic, R. (2015). The reliability of motion analysis of elite soccer players during match measured by the Tracking Motion software system. *IEEE 15th International Conference on Bioinformatics and Bioengineering.*

Redmon, J. & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. *arXiv*.

Rosen, K. H. (2004). *Matemática Discreta y sus Aplicaciones* (Quinta Edición). McGraw-Hill.

Sabirin, H., Sankoh, H. & Naito, S. (2015). Automatic Soccer Player Tracking in Single Camera with Robust Occulsion Handling Using Attribute Matching. *Online ISSN : 1745-1361 Print ISSN : 0916-8532, E98.D*.

Sahbani, B. & Adiprawita, W. (2016). Kalman Filter and Iterative-Hungarian Algorithm Implementation for Low Complexity Point Tracking as Part of FastMultiple Object Tracking System. *ICSET*.

Seo, Y., Choi, S., Kim, H. & Hong, K.-S. (1997). Where Are the Ball and Players? Soccer Game Analysis with Color Based Tracking and Image Mosaick. En *Proceedings of the 9th International Conference on Image Analysis and Processing-Volume II* (pp. 196-203). ICIAP '97. London, UK, UK: Springer-Verlag. Recuperado desde http://dl.acm.org/citation.cfm?id=646276.686879

Seo, Y. & Hong, K.-S. (2004). Multiple Soccer Players Tracking by Condensation with Occlusion Alarm Probability. *IEICE TRANSACTIONS on Information and Systems*.

Shitrit, H. B., Berclaz, J. & Fleuret, F. (2011). Tracking multiple people under global appearance constraints. *IEEE International Conference on Computer Vision*, 137-144.

Shitrit, H. B., Berclaz, J. & Fleuret, F. (2014). Multi-Commodity Network Flow for Tracking Multiple People. *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 36*(Issue 8), 1614-1627.

Siles, F. [F.] & Saborío, J. C. (2015). Parallel Spatial Segmentation for the Automated Analysis of Football. *5th IEEE International Workshop and Conference on Bioinspired Intelligence IWOBI 2015*.

Siles, F. [Francisco]. (2014). *Automated Semantic Annotation of Football Games from TV Broadcast* (Tesis doctoral, Technische Universitad Munchen).

Sitton, D. (1996). Maximum Matching in Complete Multipartite Graph. *Electronic Journal of Undergraduate Mathematics*.

Solomon, C. & Breckon, T. (2011). *Fundamentals of Digital Image Processing. A Practical Approach with Examples in Matlab* (Wiley-Blackwell, Ed.). John Wiley & Sons, Ltd.

Soomro, K., Khokhar, S. & Shah, M. (2015). Tracking When the Camera Looks Away. *ICCVW*.

Soviany, C. (2003). *Embedding Data and Task Parallelism in Image Processing Applications* (Tesis doctoral, Tu Delft).

Taixe, L. L. (2016). *Multiple object tracking with context awareness* (Tesis doctoral, Gottfired Wilhelm Leibniz Universitat Hannover).

Tanenbaum, A. S. (2006). *Operating Systems Design & Implementation*. Pearson Prentice Hall.

Technology, K. S. (2017). Kizanaro. http://www.kizanaro.com/. Recuperado desde http://www.kizanaro.com

Villalta, M. (2016). Diseño, implementación y validación de una biblioteca en paralelo híbrida CPU/GPU de algoritmos de reconocimiento de patrones para el Clúster Tará del PRIS-Lab. U.C.R.

Wire, H. (2019). Amdahls Law. Recuperado desde https://www.hpcwire.com/2015/01/22/compilers-amdahls-law-still-relevant/

WL, L., JA, T., JJ, L. & KP, M. (2013). Learning to track and identify players from broadcast sports videos. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Wood, R. (2017). Top 10 List of the World's Most Popular Sports. http://www.topendsports.com/world/lists/popular-sport/fans.htm. Recuperado desde http://www.topendsports.com/world/lists/popular-sport/fans.htm

Wu, Y. (2013). *An Introduction to Computer Vision*. Northwester University.

Xu, M., Orwell, J. & Jones, G. (2004). Tracking Football Players with Multiple Cameras. *International Conference on Image Processing (ICIP)*, 2909-2912.

Yilmaz, A., Javed, O. & Shah, M. (2006). Object Tracking: A Survey. *ACM Comput. Surv. 38*(4). doi:10.1145/1177352.1177355

Yuan, Y., Emmanuel, S. & Fang, Y. (2019, 26 de abril). Visual Object Tracking Based on Backward Model Validation. En *IEEE Transactions on Circuits and Systems for Video Technology* (Vol. 24).

Apéndice

Apéndice A

Resultados experimentales

A.1. ISSIA

Tabla A.1: Resultados de métricas de precisión con ISSIA

GTobj	SUTobj	FP	FN	TP	Recall	Precision	F1 Score
66476	74735	19.70 %	17.49 %	82.51 %	0.83	0.79	0.81

Apéndice B

Referencia de artículo

Parallelization of a Multipartite Graph Matching Algorithm for Tracking Multiple Football Players

M. Villalta and F. Siles
Pattern Recognition and Intelligent Systems Laboratory
Department of Electrical Engineering, School of Engineering
Universidad de Costa Rica, San José, Costa Rica
marco.villalta@ucr.ac.cr, francisco.siles@ucr.ac.cr

***Abstract*—This work describes the parallel methodology for a football tracking algorithm based on multipartite graphs using MPI and OpenMP. The proposed algorithm use a consumer-producer scheme to overlap the computing time of the two main procedures of the tracking algorithm: segmentation and tracking; as well a send-and-receive communication pattern to propagate the blob identities. We show how an hybrid system of data and task parallelization improves the execution time for 4K videos, achieving a speedup equal to 19.24 and a processing speed of 21.71 FPS with 128 threads.**

***Index Terms*—Parallel algorithms; Association Football; Temporal Segmentation; Tracking of Football Players**

I. Introduction

Football is a sport that has shown great popularity over the years, with an estimated 3.5 billion fans all over the world [1]. In the same way as technology has been introduced into everyday life activities, there is also a desire to incorporate it into the sports' field. Such involvement has led to the fact that in recent years, there has been an increased interest for the development of advanced computer systems that favor the automated analysis of sport's results. This kind of information is increasingly important, not only for the general viewer for entertainment purposes, but also for professional football clubs, and their coaching teams, as well as for the sport sciences community, since it will lead to perform physical assessments, fatigue detection, analysis of opponents, evaluation of tactical performance and others [2].

In order to build such automated analysis systems, one of the first steps consists of the extraction of relevant positional information of the players, which in fact requires primarily their trajectories on the field at any given time. This process of following each player to reconstruct their 3D trajectories is called player tracking. Since football is a team sport, then the player tracking stage requires detecting multiple players, finding their positions at regular intervals, and associating spatial and temporal information to extract their trajectories [3]. This task is complex due to the patterns of unpredictable movements of the players during the game, in addition to the fact that players on the same team look alike and, frequently, players find themselves struggling for ball possession, thus creating multiple occlusions. Player tracking is also affected by external factors such as environmental conditions like rain, light changes, and stadium's shadows. The ability to obtain visual quality and the accuracy of tracked targets is highly desired in a tracking system, this requires a high resolution of the input data that provide more details to the tracker. Low quality or low resolution of videos, system noise, small objects and other factors generate less precise tracking results. In case of such conditions an object would be more difficult to identify and track because the object has less information [4], [5].

The present work is part of ACE, a larger system still under development, which intends to meet the needs mentioned above. ACE is a computational platform for analyzing digital videos of football [6], which generates abstract models of the game for interpretation. This platform is implemented in a multilayer architecture, as shown in figure 1. The first four layers are related to the perception stages, while the last two are related to the semantic analysis. Since the platform was initially developed for videos taken from TV Broadcast [7], [8], [9], a *temporal segmentation* process was required, in order to select the candidate scenes (far-view scenes) where information of the players' positions can be extracted [10], [11]. The current work is focused on the lower stages: from the video acquisition to the tracking stage, bypassing the temporal segmentation, since the input video comes from the concatenation of two digital videos taken using 2 static 4K cameras placed in the stadium (see figure 4).

Given the configuration of the two 4K cameras that produce the input data to the system, and since each image from each camera has an UHD (ultra high definition) of 3840×2160, and the frame rate of $60\ Hz$, then the pixels per second (pps) to process in order to cope with the incoming information is of $995\ 328\ 000$ pps. It is therefore, necessary to consider strategies that accelerate the processing of this information. In this paper, we present a proof of concept for a parallel hybrid approach to an off-line football tracker based on multipartite graphs for the ACE platform.

II. Related Work

A study of the different types of trackers for football players is beyond the scope of this work, for that see [13]. Our scope is limited to those that present computational costs

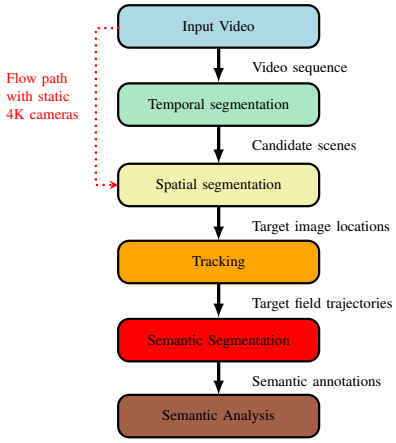


Figure 1: Layered architecture of ACE-Football. Edited from [12].

results.

Certainly, the main objective of a football player tracker is the precise identification and track of every object on the field, however, the computational cost and latency have already been considered in the professional literature. These considerations can be found since 2000 where Lefèvre et al. [14] created a method based on a fast snake obtaining a processing time of 2 seconds per frame, with a implementation on Matlab for a sequence of 100 images of 24 bits, with width and height of 384 and 288 pixels respectively. Other works show similar speed performance using Matlab as a development plataform, [15] measures 4 FPS of the motion detection stage for a particle filter tracker, [16] obtain 3 FPS for their visual object tracker, [17] 3.3 FPS in a semi supervised system and [18] for a based adaptive Kalman Filter tracker. Ma et al. [19] use fully-convolutional siamese networks for the extraction of features and the occlusion solver, their algorithm is implemented in Matlab using MatConvNet toolbox, runs at 4 FPS on a Intel Core i7-4720HQ and an NVIDIA GeForce GTX 960M GPU. Matlab is good for developing and prototyping algorithms but shows lower values of FPS on the implementations of the tracking algorithms, this is because it is a scripting language that uses a JIT compiler to translate a script to machine code.

One approach proposed for a player tracking system using a model field particle filter is Sentioscope [3]. Their algorithm is implemented in C++ on GPU using a task parallel technique running the following tasks on different threads: image acquisition, automatic exposure, light adjustment, foreground extraction and HOG calculation; while the player classification and tracking tasks execute in parallel with a gathering pattern. They mention that the system runs on a laptop with a 4 core Intel i7 CPU achieving a ratio of 14 FPS.

There are similar data association algorithms to the algorithms based on graph theory, for example [20] uses a network flow formulation as a linear programming optimization problem, their algorithm is implemented in C++ using STL. They use a 3Ghz PC and utilized a single core, obtaining a speed of 3.95 FPS for the MCNF version of the algorithm on the ISSIA database.

The image processing stages have great impact on the execution speed of trackers, therefore the parallelization of the spatial segmentation stage of our algorithm has been discussed previously on [21], where a parallel method is designed relying on multi-threading with OpenMP. They perform several tests using standard definition videos obtaining a speedup factor of 4 and efficiency of 0.1 with 8 threads. This algorithm of the parallel method was implemented in C++ with the OpenCV library. Despite of efforts made in [21], it is considered incomplete for the purposes of a tracking algorithm for being decoupled from this stage, also is an algorithm that does not scale well beyond the presented results.

It is interesting how scientific articles mention that the execution times can be improved using parallelization techniques, however they do not consider or use the distributed systems or GPGPU [18], [22], [23]. All of the related works are executed on personal computers and the FPS values presented by these works show that their implementation and resources are not sufficient to achieve a desired processing time (equal or greater than 30 FPS).

III. Implementation

A graph is a mathematical abstract representation consisting of a set of nodes, which represent objects, and a set of edges, which represent associations between object pairs, ([24]) as shown on figure 2. A graph is k-partite if they can be partitioned into k disjoint independent sets. A k-partite graph is called multipartite, typically only when $k \geq 3$ and when $k = 2$ is called bipartite [25]. In our propose method, the nodes are visualized as blobs that might or not correspond to football players and the edges as the weighted value corresponding to the similarity between blobs. The nodes encapsulate a vector of characteristics that describe the blob using their shape, chromatic information, direction and position from a kinematic model of constant speed, while the edges are the relation between those blobs from frame to frame.

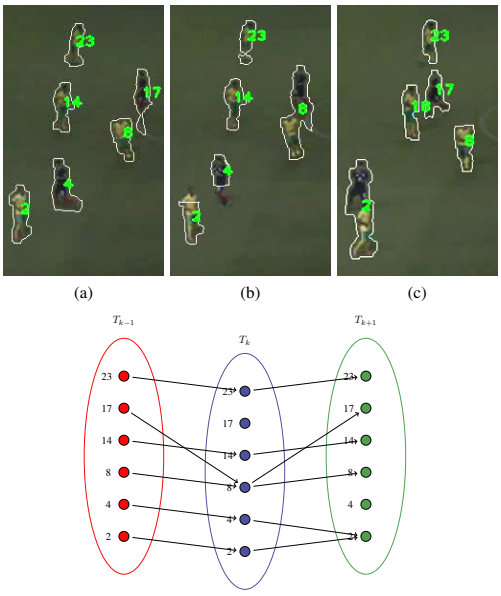


Figure 2: (a) First sequence frame, (b) Middle sequence frame, (c) Last sequence frame and a multipartite graph describing the merge/split events

Algorithm 1 Sequential Multipartite Graph Tracking algorithm

Input V Video
Output T Tracking points

procedure BATCH PROCESSING
  step $\leftarrow k$
  for $step < v$ **do**
    procedure SEGMENTATION
      Initialize G
      for $i < k$ **do**
        $f_n \leftarrow i + step$ ▷ Update first frame number of the multipartite graph
        $C \leftarrow \sigma(f_n)$ ▷ Apply segmentation function to a frame to generate a set of contours
        for $j < n$ **do**
          Generate N_j model ▷ Where N is a node
          $G_i[j] \leftarrow N_j$ ▷ Save node model on its corresponding graph
    procedure TRACKING
      if $step \neq f_f$ **then**
        ς ▷ Perform bipartite matching function between previous multipartite graph and actual multipartite graph
      $\tau(G)$ ▷ Apply tracking function
      $G_f \leftarrow G_l$

Under these circumstances we propose a parallel hybrid method to solve the tracking of multiple football players for the data association technique of multipartite graphs, relying on data and task parallelization.

Let V be a video of v frames and G a multipartite graph of k number of graphs, each one with n number of nodes. Where f_n corresponds to the processed frame number on the video V, G_f is the first graph and G_l the last graph of a multipartite graph.

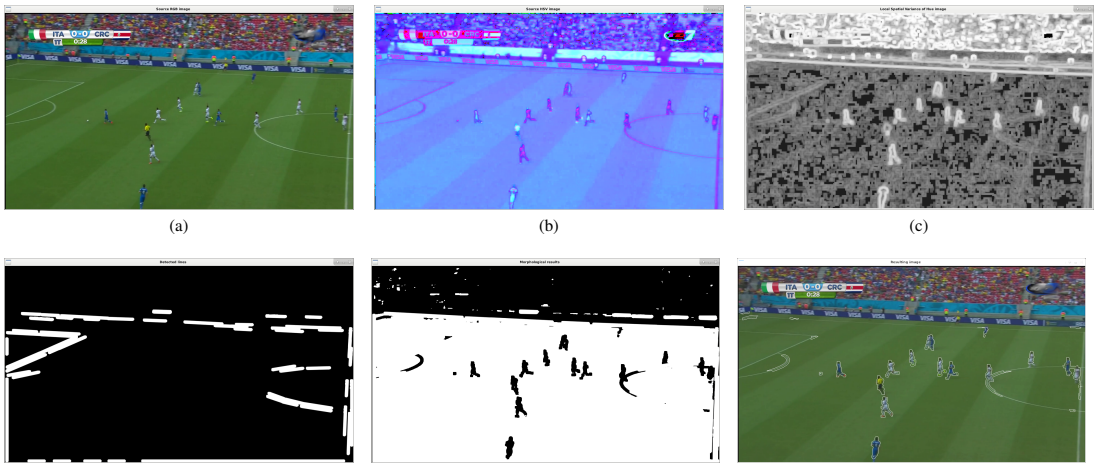
As can be seen in the algorithm 1, V is processed in groups of frames (batch mode) equals to k frames for each $step$ (which is the first processed frame on a group or batch) while $step < v$. The frame fn is read from V and is applied the segmentation function σ to generate the contours C, then each contour $c_i \in C$ is used to generate the N_j node model for the graph G_i. Once the multipartite graph has been populated with the k-graphs, a tracking function τ generates the T tracking points, which are 2D locations on the image frame as *(x,y)* coordinates. To propagate identities between consecutive multipartite graphs a bipartite matching function ς is performed between the adjacent graphs.

Despite the sequential behavior exhibited by the algorithm 1, it is possible to overlap its two main procedures with a *consumer-producer* scheme where the even processes perform the segmentation function σ and generate the multiparite graphs that the odd processes use on the tracking function τ. Following the same notation as the algorithm 1, it is assumed that there are p processes with a corresponding identification number p_{id}, μ is the maximum number of iterations for those processes, f_a is the start frame for each process, f_o is the stop frame for each process, f_0 the first frame of V, where G_f and G_l are the first and last graph for the batch of process p_{id}.

The spatial segmentation algorithm used is an improved version of [21] for tracking purposes, whose results are shown in the figure 3, which includes the following functions:

1) HSV color conversion
2) Normalized Local Spatial Variance for the Hue and Value components
3) Identification of green regions
4) Edge detection
5) Removal of field lines with Hough transformation.
6) Morphological transformations of erosion and dilation.
7) Discard spurious regions and blob filtering using a circularity criterion.
8) Blob model creation within the multipartite graph.

This set of segmentation and filtering functions are observed in the figure 3, where it starts with a raw frame that is then processed to obtain the contours of the players, we use OpenMP as the method of parallelization on each of these functions.



(a) (b) (c)

(d) (e) (f)

Figure 3: Samples from segmentation procedure: (a) Source frame, (b) HSV frame conversion, (c) Local spatial variance for the Hue component, (d) detected lines, (e) Homogeneous regions with line filtering applied, (f) Resulting segmentated contours

Algorithm 2 Parallel Multipartite Graph Tracking algorithm

Input V Video
Output T Tracking points

1: $\mu \leftarrow v/((p/2) * k)$
2: **procedure** BATCH PROCESSING
3: **for** $i <= \mu$ **do** ▷ Loop until reach max iterations
▷ Compute first and last frame of the multipartite graph for the even and odd processes
4: **if** p_{id} % 2 **then**
5: $f_a \leftarrow k * p_{id} + i + f_a$
6: $f_o \leftarrow f_a + k - 1$
7: **else**
8: $f_a \leftarrow k * p_{id} + k * p * i + f_a$
9: $f_o \leftarrow f_a + k - 1$
10: **procedure** PARALLEL SEGMENTATION
11: **if** p_{id} % 2 **then**
12: **for** $j < k$ **do**
13: $f_n \leftarrow i + f_a$ ▷ Current frame number
14: $C \leftarrow \sigma(f_n)$ ▷ Apply segmentation function to current frame to find contours
15: **for** $j < n$ **do** ▷ Sweep the contours to generate the node models
16: Generate N_j model
17: $g_i[j] \leftarrow N_j$
18: Send multipartite graph G from process
19: p_i to process p_{id+1}

Once, each blob model is created and assigned as a node on the corresponding graph, the tracking algorithm performs the following functions:

1) Pruning the multipartite graph based on the relation between nodes that includes the comparison of static and dynamic descriptors such as: histogram, position, direction, shape, area and speed.
2) Building a coincidence matrix.
3) Forward analysis of borders for merge/occlusion events.
4) Backward analysis of borders for split events.
5) Propagation of blob identities.

The algorithm 2 describes the segmentation and tracking procedures on a batch mode equals to the multipartite graph size. From lines 4 through 9 show how the distribution of frames is determined for each thread and their corresponding start and ending positions within the video. From lines 18 to 23 the consumer-producer schema is executed, where the parallel segmentation sends multipartite graphs, received by the parallel tracker allowing the overlap of those computational tasks. Another communication pattern between processes is from lines 27 to 31 and from lines 35 to 48 where the send and receive actions are implemented between odd adjacent processes to transfer a graph, and propagate the blob identities. Between lines 49 and 58 we deal with the special case of two processes: one producer and one consumer.

The generation of the concatenated 4K input video, was performed using an algorithm based on geometric transformations that finds a function which transforms the points in a trapezoid to points of a rectangle in a continuous way. In order to obtain more abundant and detailed

```
20:  procedure PARALLEL TRACKER
21:      if p_id % 2 = 1 then
22:          Receive multipartite graph G from process
23:          p_{id-1}
24:          if p ≠ 2 then ▷ General case of more than 2 processes
25:              if f_a ≠ f_f ∨ i == μ then
26:                  if p_id == 1 then
27:                      Receive graph G_f
28:                      from process p_{id-1}
29:                  else
30:                      Receive graph G_f
31:                      from process p_{id-2}
32:                  ς(G_f) ▷ Bipartite matching function to propagate identities
33:              τ(G) ▷ Apply tracking function
34:              if i == 1 then ▷ First iteration
35:                  if p_id < p − 1 then
36:                      Send graph G of frame f_o
37:                      from process p_id to p_{id+2}
38:              if i ≠ 1 ∨ μ then ▷ Cases between first and last iteration
39:                  if pid < p − 1 then
40:                      Send graph G of frame f_o
41:                      from process p_id to p_{id+2}
42:                  else
43:                      Send graph G of frame f_o
44:                      from process p_id to p_1
45:              else ▷ Last iteration
46:                  if pid < p − 1 then
47:                      Send graph G of frame f_o
48:                      from process p_id to p_{id+2}
49:          else ▷ Special case of 2 processes
50:              if fg ≠ f_a ∪ i ≠ μ then
51:                  ς(fg)
52:                  τ(G)
53:                  G_l ← G(f_o)
54:              else
55:                  if i = μ − 1 then
56:                      ς(G_f)
57:                      τ(G)
58:                  G_l ← G(f_o)
59:
60:
61:
```

Table I: Execution times for sequential algorithm

Resolution	Run time (s)	Segmentation time (s)	Track time (s)
4K	2267	2229	38
FHD	542	530	12
HD	286	283	3

Table II: HW specifications of the cluster TARÁ

Node	Quantity	Processor	Ram	HD
Master	1	Intel Xeon E5-2650	62 GB	500 GB
Compute	4	Intel Xeon E5-2650	251 GB	500 GB
Storage	4	Intel Xeon E5-2650	62 GB	40 TB

information on the players while simultaneously avoiding the loss of vital information, this camera configuration was used. An example of the resulting panorama image is shown in figure 4.

The implementation of the segmentation and tracking algorithm was programmed in C++, and have used software tools/libraries consisting of: GCC 4.9.2, openmpi 2.1.3, opencv 2.4.13.5, and the boost library 1.67.0.

IV. Results and Discussion

The experiments were ran on the cluster TARÁ from the PRIS-Lab, the hardware specifications are shown on table II. It has three types of nodes, the master for overall control, the processing nodes, and the storage nodes. All nodes are connected to a private Gigabit Ethernet internal network by means of four network interfaces operating in a bonding mode to maximize the available bandwidth. This network topology allowed Lustre, the file system available in the 4 storage nodes, to be in advantage, as this file system has high performance capabilities, and allows to have better reading performance in parallel tasks [26], [27]. Each processing node has 2 Intel Xeon E5-2650 running at 2.0 GHz for a total of 64 cores or 128 threads. For each MPI process 2 OpenMP threads were enabled following a processor architecture affinity criterion, from a previous work [28] it was determined that this was the optimal configuration for the cluster.

The speedup was obtained as $S = T_s/T_p$, and the efficiency as $E = S/p$, where T_s is the sequential running time, T_p the parallel running and p as the number of threads. The results of both metrics are shown on figures 5 and 6 for a 4K video sequence, with a resolution of 3648×512, and

Table III: Results for a 4K video

Threads	Run time (s)	Speedup	Efficiency	FPS	Track time (s)
128	117.85	19.24	0.15	21.71	2.07
64	176.69	12.83	0.20	14.48	2.96
32	218.51	14.45	0.45	12.44	3.89
16	426.27	7.41	0.46	6.38	5.20
8	519.44	6.23	0.78	5.39	8.05
4	1215.93	2.69	0.67	2.33	16.91
2	2190.06	1.49	0.75	1.30	37.69
1	2267.00	1.00	1.00	1.27	38.00

Figure 4: Input video frame.

Table IV: Multipartite graph size effect

MPG size	Threads	Run time (s)	Speedup	Efficiency	FPS	Track time (s)
20	128	117.85	19.24	0.15	21.71	2.07
10	128	164.27	13.80	0.11	15.58	14.19
5	128	217.96	10.40	0.08	11.74	15.19

Table V: Results for different video resolutions

Resolution	Threads	Run time (s)	Speedup	Efficiency	FPS	Track time (s)
4K	128	117.85	19.24	0.15	21.71	2.07
FHD	128	31.10	24.15	0.19	82.28	2.00
HD	128	17.71	23.99	0.19	144.47	1.95

2877 frames of length, the results are shown also in the table III with the FPS and execution times. We inspect the effect of different sizes of multipartite graphs and dimensions of videos on the speedup and efficiency in tables IV and V for 128 fixed number of threads.

From tables I, III, IV and V it is observed that most of the execution time is spent on the segmentation part of the algorithm, as being affected in particular by the detection of contours that occupies 36.66% of the execution time from this section.

From table III it is observed that the data parallelization, as being distributed equally among the even processes, and the task parallelization of the segmentation and tracking procedures improves the speedup and frames per second. If the relation between the odd and even threads is maintained, the tracking time remains between 1.22% and 1.78% of the total execution time, this is because the minimization of the multipartite graphs uses computationally simple functions and the propagation of identities does not consume many

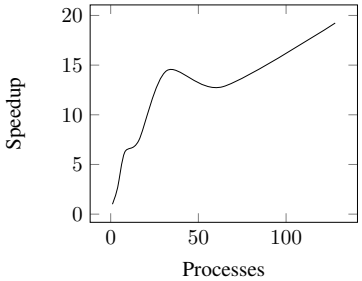


Figure 5: Speedup for a 4K video.

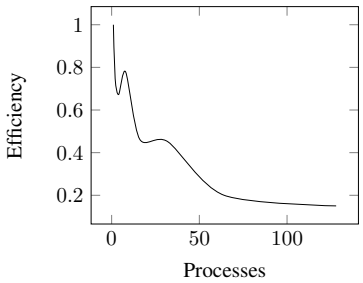


Figure 6: Efficiency for a 4K video.

resources. The growth of acceleration based on the size of the multipartite graph is probably due to a smaller amount of communications between adjacent nodes under the cost of a greater use of memory as shown in table IV. The information of table III, figure 5 and figure 6 shows how consistently there is a speedup, but with a penalty in the efficiency due to the sequential sections of the algorithm.

V. Conclusions and Future Work

In this paper, we deal with the natural sequential process of tracking football players, exploring the hybrid model formed by the combination of MPI and OpenMP parallelism.

In the professional literature of football tracking algorithms there are no references of parallelization for distributed memory systems. Although the complexity of algorithm 2 is greater than algorithm 1, the benefits associated with the execution times lead us to the conclusion that a parallel hybrid system must be considered for the conditions of this application with an algorithm that takes advantage of such capabilities. This is the main contribution of our work.

As shown in figure 5, our algorithm has good prospects to scale to a greater number of processes, however, caution should be taken with the graph size in terms of memory consumption, despite of showing better results by increasing its size according to table IV. The speedup appears to be linear inside a node but when more nodes are used there is a penalty for network communication. The efficiency values also show better results than others methods for a more complex version of the segmentation algorithm.

As demonstrated by the results shown on tables I, III, V and IV, the segmentation section is the bottle neck of the

algorithm requiring heavy computational power. To achieve real-time results, several challenges must be faced in both hardware and software. In relation to hardware, it is necessary to have capable resources to perform fast parallel I/O and communications. For our setup, the speed of the network must be improved by changing the GigaEthernet interfaces for Infiniband interfaces. For future work we expect better results, higher FPS values, with a GPU version of the segmentation and tracking algorithm in conjunction with the distributed model. The proposed tracking algorithm can be used to improve the results of other applications such as cell tracking, for this it is necessary to modify the segmentation procedure partially or totally depending on the characteristics of the input data, to extract the visual features [29]. It is also important to explore the use of OpenMP in accelerators such as XeonPhi cards for this type of algorithms.

ACKNOWLEDGMENTS

The present paper is a partial result of a trans-disciplinary research project entitled *Design, implementation and validation of a computational platform for the semantic annotation of football from digital videos*, registered in the Research Vice presidency at the UCR with id *838-B6-751*, and the Human Movement Research Center (CIMOHU) as the academic unit base. Also the Department of the Electrical Engineering supports this research project.

A special thanks to the Postgraduate Program in Electrical Engineering of the Postgraduate Studies System, UCR for the help during my Master Program. Also, to the members of the PRIS-Lab for their friendship and support during my work.

REFERENCES

[1] R. Wood, "Top 10 list of the world's most popular sports," http://www.topendsports.com/world/lists/popular-sport/fans.htm, May. 2017. [Online]. Available: http://www.topendsports.com/world/lists/popular-sport/fans.htm

[2] R. Radakovic, M. Dopsaj, and R. Vulovic, "The reliability of motion analysis of elite soccer players during match measured by the tracking motion software system," *IEEE 15th International Conference on Bioinformatics and Bioengineering*, Nov. 2015.

[3] S. Baysal and P. Duygulu, "Sentioscope: A soccer player tracking system using model field particles," *IEEE Transactions on Circuits and Aystems for Video Technology*, vol. Vol. 26, no. No. 7, 2016.

[4] Y. Habibi, D. R. Sulistyaningrum, and B. Setiyono, "A new algorithm for small object tracking based on super-resolution technique," *AIP Conference Proceedings*, vol. 1867, no. 1, p. 020024, 2017. [Online]. Available: https://aip.scitation.org/doi/abs/10.1063/1.4994427

[5] P. Figueroa, N. Leite, and R. M. L. Barros, "Tracking soccer players using the graph representation," in *Proceedings of the 17th International Conference on Pattern Recognition*, 2004, pp. 787–790.

[6] F. Siles, "Ace-football analysing football from tv broadcasting." *24th German Soccer Conference DVS FuSSball in Lehre und Forshung, Weiler*, 2013.

[7] R. C. Quesada and F. S. Canales, "Evaluation of different histogram distances for temporal segmentation in digital videos of football matches from tv broadcast," *2017 International Conference and Workshop on Bioinspired Intelligence*, 2017.

[8] S. C. Ramírez and F. S. Canales, "Deceived bilateral filter for improving the classification of football players from tv broadcast," *Bio-inspired Intelligence (IWOBI), 2014 International Work Conference on*, pp. 98 – 105, 2014.

[9] F. Siles, "Shot classification for association football from tv broadcasting," *Applied Machine Intelligence and Informatics (SAMI), IEEE 12th International Symposium on*, 2014.

[10] R. C. Quesada, S. C. Ramírez, and F. Siles, "Improving the temporal segmentation in digital videos using the deceived bilateral filter," *IEEE 36th Central American and Panama Convention(CONCAPAN)*, 2016.

[11] F. Siles, "Temporal segmentation of association football from tv broadcast." *INES 2013*, 2013.

[12] F. Siles Canales, "Automated semantic annotation of football games from tv broadcast," Ph.D. dissertation, Uni München, 2014.

[13] M. Manafifard, H. Ebadi, and H. A. Moghaddam, "A survey on player tracking in soccer videos," *Computer Vision and Image Understanding*, 2017.

[14] S. Lefèvre, C. Fluck, B. Maillard, and N. Vincent, "A fast snake-based method to track football players," *IAPR International Workshop on Machine Vision Applications*, 2000.

[15] R. C. H. de Vos and W. Brink, "Combining motion detection and hierarchical particle filter tracking in a multi-player sports environment," *Proceedings of the 20th Symposium of the Pattern Recognition Association of South Africa*, 2009.

[16] Y. Yuan, S. Emmanuel, and Y. Fang, "Visual object tracking based on backward model validation," in *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, 2014.

[17] R. Martín and J. M. Martínez, "A semi-supervised system for players detection and tracking in multi-camera soccer videos," *Multimedia Tools Appl.*, vol. 73, no. 3, pp. 1617–1642, Dec. 2014. [Online]. Available: http://dx.doi.org/10.1007/s11042-013-1659-6

[18] N. Najafzadeh, M. Fotouhi, and S. Kasaci, "Multiple soccer players tracking," *2015 International Symposium on Artificial Intelligence and Signal Processing*, 2015.

[19] Y. Ma, S. Feng, and Y. Wang, "Fully-convolutional siamese networks for football player tracking," *2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS)*, 2018.

[20] H. B. Shitrit, J. Berclaz, and F. Fleuret, "Multi-commodity network flow for tracking multiple people," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. Vol. 36, no. Issue 8, pp. 1614 – 1627, Aug. 2014.

[21] F. Siles and J. C. Saborío, "Parallel spatial segmentation for the automated analysis of football," *5th IEEE International Workshop and Conference on Bioinspired Intelligence IWOBI 2015*, 2015.

[22] R. Hoseinnezhad, B.-N. Vo, B.-T. Vo, and D. Suter, "Visual tracking of numerous targets via multi-bernoulli filtering of image data," *Pattern Recogn.*, vol. 45, no. 10, pp. 3625–3635, Oct. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.patcog.2012.04.004

[23] H. Morimitsu, R. M. Cesar, and I. Block, "Attributed graphs for tracking multiple objects in structured sports videos," *IEEE Iternational Conference on Computer Vision*, 2015.

[24] D. Sitton, "Maximum matching in complete multipartite graph," *Electronic Journal of Undergraduate Mathematics*, 1996.

[25] C. A. Phillips, "Multipartite graph algorithms for the analysis of heterogeneous data," Ph.D. dissertation, University of Tennesse, 2015.

[26] T. Zhao, V. March, and S. Dong, "Evaluation of a performance model of lustre file system," *2010 Fifth Annual ChinaGrid Conference*, 2010.

[27] S. Jian, L. Zhan-huai, and Z. Xiao, "The performance optimization of lustre file system," *2012 7th International Conference on Computer Science and Education*, 2012.

[28] M. Villalta, "Diseño, implementación y validación de una biblioteca en paralelo híbrida cpu/gpu de algoritmos de reconocimiento de patrones para el clúster tará del pris-lab," U.C.R., Noviembre 2016.

[29] A. M. Zuniga, S. Q. Barrantes, and F. S. Canales, "M-phase feature extraction algorithm for phenotype classification from cancer brightfield microscopy," 2018.

Apéndice C

Código fuente

C.1. MPGT secuencial

```
1  /**
2   * This is a sequential version of the
3   * Multipartite Graph Tracker MPGT
4   **/
5
6  //includes
7  #include <iostream>
8  #include <fstream>
9  #include <string>
10 #include <cstdlib>
11 #include <vector>
12 #include <queue>
13 #include <opencv2/opencv.hpp>
14 #include <getopt.h>
15 #include "PrisSerial.h"
16 #include "Defs.h"
17 #include "VideoManager.h"
18 #include "Structs.h"
19 #include "PlayerBlob.h"
20 #include "Segmentator.h"
21 #include "MultiPartiteGraph.h"
22 #include "YamlParser.h"
23 #include "Timer.h"
24
25 //namespaces
```

```
26 using namespace std;
27 using namespace cv;
28
29 //global vars
30 std::string input_file = "input_file.dv";
31 std::string output_file = "video_file.avi";
32 std::string track_file = "tracking_file.yaml";
33 std::string config_file = "config.yaml";
34 std::string init_file = "init.yaml";
35 vector< cv::Point > centers;
36 vector< TrackPoint > init_tpoints;
37 bool create_video = false;
38 bool enable_tracking = false;
39 bool draw_centers = false;
40 bool draw_ids = false;
41 bool load_init_data = false;
42
43 //function prototypes
44 void PrintHelp(void);
45 void ProcessArgs(int argc, char** argv);
46
47 //functions
48 int main(int argc, char *argv[])
49 {
50
51    /// Init program: read args & video properties
52    ProcessArgs(argc, argv);
53    cout << "Loading video file ..." << endl;
54
55    VideoManager fh; //Video player object-file handler
56    fh.SimpleVideoLoader(input_file);
57    int isize = fh.bufVideoInfo.frames;
58
59    double dWidth = fh.bufVideoInfo.width; //get the width of frames of the video
```

```
60    double dHeight = fh.bufVideoInfo.height; //get the height of frames of the
          ↪ video
61    double fps = fh.bufVideoInfo.fps; //get the frames per seconds of the video
62    double tframes = fh.bufVideoInfo.frames; //get the total frames
63
64    cout << "Frame per seconds : " << fps << endl;
65    cout << "Total frames in video : " << tframes << endl;
66    cout << "Frame Size = " << dWidth << "x" << dHeight << endl;
67
68    cv::Mat iframe, oframe;
69
70    cout << "Starting segmentation and tracking..." << endl;
71    Timer tm,seg_cronometer,tracking_cronometer,writing_cronometer;
72
73    tm.Start("MPGT segmentation and tracking");
74    seg_cronometer.CrnStart("Segmentation");
75    tracking_cronometer.CrnStart("Tracking");
76    if(create_video)
77       writing_cronometer.CrnStart("Wrinting");
78    seg_cronometer.CrnPause();
79    tracking_cronometer.CrnPause();
80    if(create_video)
81       writing_cronometer.CrnPause();
82
83    YamlParser yp;
84    yp.LoadConfig(config_file);
85    yp.segparms["video_height"]=dHeight;
86    yp.segparms["video_width"]=dWidth;
87    yp.InitResultWriter(track_file);
88    yp.StoreHeaderInfo(input_file, fh.bufVideoInfo);
89    if(load_init_data)
90       init_tpoints = yp.LoadTrackingFile(init_file, "non");
91
92    ///Segmentator object and operation mode;
```

```
93    Segmentator sg(yp.segparms);
94
95    if(create_video)
96       fh.StartVideoSaver(output_file);
97    ///Run on each frame on a batch mode equal to the graph window until stop frame
          ↪  % window size to avoid seg faults
98    Graph bpg_first_graph;
99    OclusionMemoir OMemory; ///Oclusion event/node memory
100   AbductionMemoir AMemory; ///Abduction event/node memory
101   OMemory.InitMemoir(yp.mpgtparms,yp.weights);
102   AMemory.InitMemoir(yp.mpgtparms,yp.weights);
103   int stop_frame_wsize = (int)yp.stop_frame - ((int)yp.stop_frame % yp.
          ↪ graph_window);
104   for(int step=yp.start_frame;step<isize && step<= stop_frame_wsize;step=step+yp.
          ↪ graph_window){
105
106      MultiPartiteGraph mpgo(yp.graph_window);
107      mpgo.initial_frame=step;
108      mpgo.last_frame=step+yp.graph_window-1;
109      D(cout << "Working on block from " << mpgo.initial_frame << " to " << mpgo.
             ↪ last_frame << endl;)
110
111      ///**************First step: Segmentation**************
112      for(int i=0; i<yp.graph_window;i++) {
113
114         seg_cronometer.CrnResume();
115         int frame_number = i+step;
116         D(cout << "Extracting blobs from frame " << frame_number << endl;)
117         iframe = fh.GetFrame(frame_number);
118         if(iframe.empty())
119            break;
120
121         //cv::normalize( iframe, iframe, 0, 1, cv::NORM_MINMAX, CV_32FC3, cv::Mat
               ↪ () );
```

```
122         vector< vector< cv::Point > > contornos_jugadores = sg.Segment(iframe);
123
124         ///Adquirir los descriptores para cada blob y guardarlos en los nodos
                ↪ correspondientes
125         for(unsigned int j=0;j<contornos_jugadores.size();j++){
126            PlayerBlob blob;
127            blob.FillModel(frame_number,j,contornos_jugadores[j],iframe);
128            if(load_init_data)
129               blob.AssignInitTag(init_tpoints,yp.init_tkp_threshold);
130            if((draw_centers==true) || (yp.drawparms["draw_trayectories"]))
131               centers.push_back(blob.centroid);
132
133            ///Quitar los blobs que estan fuera de los limites superiores e
                   ↪ inferiores
134            if(yp.segparms["enable_boundary_blob_remover"]){
135               double high_limit = (double)dHeight*(double)yp.segparms["
                      ↪ lower_boundary"]/100.0;
136               double low_limit = (double)dHeight*(double)yp.segparms["
                      ↪ upper_boundary"]/100.0;
137               //cout << "Blob with id " << blob.blob_id << " on frame " <<
                      ↪ frame_number << " at [ " << blob.centroid.x << " , " << blob
                      ↪ .centroid.y
138               // << " ] inspected" << " lower limit = " <<low_limit << " higher
                      ↪ limit " << high_limit << endl;
139               if((blob.centroid.y <= low_limit) &&
140                  (blob.centroid.y >= high_limit) ){
141                  mpgo.InsertNode(frame_number,blob);
142               }
143            }
144
145            else
146               mpgo.InsertNode(frame_number,blob);
147            //cout << "Mostrando info de blob " << blob.blob_id << " en el frame "
                   ↪  << blob.blob_frame_number << endl;
```

```
148             //imshow("Contorno de jugador",blob.contourRegion);
149          }
150           seg_cronometer.CrnPause();
151       }//for loop within a multipartite graph
152
153       ///**************Second step: Tracking**************
154       if(enable_tracking){
155          tracking_cronometer.CrnResume();
156          if(step != yp.start_frame){///To exclude first frame of sequence
157             //mpgo.SolveBorderBPG(yp.weights,step,bpg_first_graph);
158             mpgo.SetOclusionMemories(OMemory.vector_memoir);
159             mpgo.SetAbductionMemories(AMemory.vector_memoir);
160             mpgo.SolveExtendedBorderBPG(yp.weights,yp.mpgtparms,step,
                    ↪ bpg_first_graph);
161             bpg_first_graph.clear();
162             OMemory.vector_memoir.clear();
163             AMemory.vector_memoir.clear();
164          }
165          //if(not yp.segparms["enable_extended_finder"])
166          mpgo.MVMPAssignFromFirstGraph(yp.weights,step);
167          mpgo.MVExtendedSimpleSolver(yp.weights,yp.mpgtparms,step); ///Tracking
                ↪ function
168
169          int last_frame_number_from_window = step+yp.graph_window-1;
170          //cout << "bpg first graph = " << last_frame_number_from_window << endl;
171          bpg_first_graph = mpgo.GetGraph(last_frame_number_from_window);
172          OMemory.vector_memoir = mpgo.GetOclusionMemories();
173          AMemory.vector_memoir = mpgo.GetAbductionMemories();
174          tracking_cronometer.CrnPause();
175
176          for(int gi=0; gi<yp.graph_window;gi++) {
177             int frame_number = gi+step;
178             yp.StoreFrameResult(mpgo.GetGraph(frame_number));
179          }
```

```
180       }
181
182       ///**************Third step: Draw results**************
183       if(create_video){
184       writing_cronometer.CrnResume();
185
186       ///Dibujar los contornos y los ids para cada frame del multigrafo
187       for(int gi=0; gi<yp.graph_window;gi++) {
188
189          int frame_number = gi+step;
190          D(cout << "Drawing results for frame " << frame_number << endl;)
191          iframe = fh.GetFrame(frame_number);
192          if(iframe.empty())
193             break;
194
195          vector< vector< cv::Point > > contornos_jugadores;
196          vector< Node > vectornode;
197
198          Graph solved_graph = mpgo.GetGraph(frame_number);
199
200          ///Extract the contours from the graph nodes
201          for(unsigned int ni=0;ni<solved_graph.size();ni++){ //ni node id
202             Node blobnode = mpgo.GetNode(frame_number,ni);
203             if(yp.drawparms["enable_boundary_blob_remover"]){
204                double high_limit = (double)dHeight*(double)yp.drawparms["
                      ↪ lower_boundary"]/100.0;
205                double low_limit = (double)dHeight*(double)yp.drawparms["
                      ↪ upper_boundary"]/100.0;
206
207                if((blobnode.centroid.y <= low_limit) &&
208                   (blobnode.centroid.y >= high_limit) ){
209                      contornos_jugadores.push_back(blobnode.contorno);
210                      vectornode.push_back(blobnode);
211                }
```

```
212             }
213             else{
214                 contornos_jugadores.push_back(blobnode.contorno);
215                 vectornode.push_back(blobnode);
216             }
217         }//for loop to sweep all nodes within a graph
218
219         cv::Mat contornos_graf = DrawContour( contornos_jugadores, iframe.rows,
                ↪ iframe.cols );
220         if((draw_centers==true) || (yp.drawparms["draw_trayectories"])){
221             D(cout << "Drawing trayectories..." << endl;)
222             for(unsigned int j=0;j<centers.size();j++)
223                 circle( contornos_graf, centers[j], 2, Scalar(255,255,255), -1, 8,
                        ↪ 0 );
224         }
225         contornos_graf = ConvertColor( contornos_graf, PRIS_GRY2RGB );
226         cv::normalize( iframe, iframe, 0, 1, cv::NORM_MINMAX, CV_32FC3, cv::Mat()
                ↪ );
227         cv::Mat oframe;
228         if(yp.drawparms["draw_contours"]){
229             oframe = Add( iframe, contornos_graf );
230             oframe = Normalize( oframe, PRIS_8UC1 );
231             if(oframe.empty()){
232                 cout << "Frame de salida vacio" << endl;
233                 break;
234             }
235         }
236         else{
237             oframe = iframe.clone();
238             oframe = Normalize( oframe, PRIS_8UC1 );
239         }
240         if(yp.drawparms["draw_bounding_box"]){
241             vector<cv::Rect> boundRect( contornos_jugadores.size() );
242             for( unsigned int i = 0; i < contornos_jugadores.size(); i++ ){
```

```
243                 boundRect[i] = boundingRect( Mat(contornos_jugadores[i]) );
244                 rectangle( oframe, boundRect[i], Scalar(200,200,200), 1, 8, 0 );
245              }
246           }
247
248           ///Dibujar los blob ids/labels
249           if(draw_ids==true || yp.drawparms["draw_ids"]==true){
250              for(unsigned int vn = 0; vn < vectornode.size(); vn++){
251                 std::string sid;
252                 if(yp.drawparms["draw_blob_ids_also"])
253                    sid = std::to_string(vectornode[vn].player_tag) + "/" + std::
                       ↪ to_string(vectornode[vn].blob_id);
254                 else
255                    sid = std::to_string(vectornode[vn].player_tag);
256
257                 putText(oframe, sid, vectornode[vn].centroid, FONT_HERSHEY_PLAIN, (
                       ↪ double)yp.drawparms["font_scale"], CV_RGB(0,255,0), 2.0);
258              }
259           }
260
261           //imshow("Video de salida",oframe); waitKey();
262           if(create_video)
263              fh.SimpleFrameSaver(oframe);
264        }
265        writing_cronometer.CrnPause();
266
267     }//for loop to sweep all frames in a batch mode
268     }
269     tm.Stop();
270     seg_cronometer.CrnStop();
271     tracking_cronometer.CrnStop();
272     if(create_video)
273        writing_cronometer.CrnStop();
274     cout << "FPS = " << (yp.stop_frame-yp.start_frame)/tm.accum << endl;
```

```
275
276     yp.CloseResultWriter();
277
278
279     return 0;
280
281 }
282
283 ///**********************Funcion que imprime ayuda
284 void PrintHelp(void)
285 {
286     std::cout <<
287           "\n Arguments: \n"
288           "-i <ifile>: Input video file\n"
289           "-o <ofile>: Output video file\n"
290           "-t <tfile>: Track file to store tracking information\n"
291           "-c <cfile>: Configuration file\n"
292           "-l <lfile>: Initialization data file\n"
293           "-C: Draw blob trayectories \n"
294           "-I: Draw blob ids \n"
295           "--help: Show help\n";
296
297     exit(1);
298 }
299
300 ///**********************Funcion que procesa argumentos
301 void ProcessArgs(int argc, char** argv)
302 {
303
304     //if (argc < 4)
305     // PrintHelp();
306
307     const char* const short_opts = "i:o:t:c:l:CIh";
308     const option long_opts[] = {
```

```
309         {"input", required_argument, nullptr, 'i'},
310         {"output", optional_argument, nullptr, 'o'},
311         {"track", optional_argument, nullptr, 't'},
312         {"config", optional_argument, nullptr, 'c'},
313         {"load", optional_argument, nullptr, 'l'},
314         {"centers", no_argument, nullptr, 'C'},
315         {"ids", no_argument, nullptr, 'I'},
316         {"help", no_argument, nullptr, 'h'},
317         {nullptr, 0, nullptr, 0}
318     };
319
320     while (true)
321     {
322        const auto opt = getopt_long(argc, argv, short_opts, long_opts, nullptr);
323
324        if (-1 == opt)
325           break;
326
327        switch (opt)
328        {
329        case 'i':
330           input_file = std::string(optarg);
331           std::cout << "Input video file is: " << input_file << std::endl;
332           break;
333
334
335        case 'o':
336           output_file = std::string(optarg);
337           std::cout << "Output video file set to: " << output_file << std::endl;
338           create_video=true;
339           break;
340
341        case 't':
342           track_file = std::string(optarg);
```

```
343         std::cout << "Tracking file set to: " << track_file << std::endl;
344         enable_tracking=true;
345         break;
346 
347       case 'c':
348         config_file = std::string(optarg);
349         std::cout << "Configuration file set to: " << config_file << std::endl;
350         break;
351 
352       case 'l':
353         init_file = std::string(optarg);
354         std::cout << "Initialization file set to: " << init_file << std::endl;
355         load_init_data = true;
356         break;
357 
358       case 'C':
359         draw_centers = true;
360         std::cout << "Drawing segmented trayectories " << std::endl;
361         break;
362 
363       case 'I':
364         draw_ids = true;
365         std::cout << "Drawing blob ids " << std::endl;
366         break;
367 
368       case 'h': // -h or --help
369       case '?': // Unrecognized option
370       default:
371         PrintHelp();
372         break;
373       }
374 
375   }
376 
```

```
377 }
```

C.2. MPGT paralelo

```
1  /**
2   * This is a parallel version of the
3   * Multipartite Graph Tracker MPGT
4   **/
5
6  //includes
7  #include <iostream>
8  #include <fstream>
9  #include <string>
10 #include <cstdlib>
11 #include <vector>
12 #include <queue>
13 #include <opencv2/opencv.hpp>
14 #include <getopt.h>
15 #include "PrisSerial.h"
16 #include "Defs.h"
17 #include "VideoManager.h"
18 #include "Structs.h"
19 #include "PlayerBlob.h"
20 #include "Segmentator.h"
21 #include "MultiPartiteGraph.h"
22 #include "YamlParser.h"
23 #include "Timer.h"
24 #include "Comunicator.h"
25
26 //namespaces
27 using namespace std;
28 using namespace cv;
29
30 //global vars
```

```
31 std::string input_file = "input_file.dv";
32 std::string output_file = "video_file.avi";
33 std::string track_file = "tracking_file.yaml";
34 std::string config_file = "config.yaml";
35 std::string init_file = "init.yaml";
36 vector< cv::Point > centers;
37 vector< TrackPoint > init_tpoints;
38 bool create_video = false;
39 bool enable_tracking = false;
40 bool draw_centers = false;
41 bool draw_ids = false;
42 bool load_init_data = false;
43 int rid;
44 int nprocs;
45
46 //function prototypes
47 void PrintHelp(void);
48 void ProcessArgs(int argc, char** argv);
49
50 //functions
51 int main(int argc, char *argv[])
52 {
53
54    /// Init program: read args & video properties
55    boost::mpi::environment boost_env;
56    boost::mpi::communicator world_MPI_boost;
57    Comunicator com;
58    com.Start(&world_MPI_boost);
59    rid = com.rid;
60    nprocs = com.world_size;
61
62    if((nprocs%2)==1){
63       cout << "This program runs only with an even number of processes" << endl;
64       com.Stop();
```

```
65        boost_env.~environment();
66        //boost_env.abort(1);
67        exit(0);
68    }
69
70    ProcessArgs(argc, argv);
71    if(rid==MASTER+1)
72        cout << "Loading video file ..." << endl;
73
74    VideoManager fh; //Video player object-file handler
75    fh.SimpleVideoLoader(input_file);
76    int isize = fh.bufVideoInfo.frames;
77
78    double dWidth = fh.bufVideoInfo.width; //get the width of frames of the video
79    double dHeight = fh.bufVideoInfo.height; //get the height of frames of the
          ↪ video
80    double fps = fh.bufVideoInfo.fps; //get the frames per seconds of the video
81    double tframes = fh.bufVideoInfo.frames; //get the total frames
82
83    if(rid==MASTER+1){
84        cout << "Frame per seconds : " << fps << endl;
85        cout << "Total frames in video : " << tframes << endl;
86        cout << "Frame Size = " << dWidth << "x" << dHeight << endl;
87    }
88
89    cv::Mat iframe, oframe;
90
91    if(rid==MASTER+1)
92        cout << "Starting segmentation and tracking..." << endl;
93
94    Timer tm;
95    Timer tracking_cronometer;
96    Timer writing_cronometer;
97    Timer seg_cronometer;
```

```
98    Timer com1_cronometer, com2_cronometer;
99
100   if(rid==MASTER+1){
101      tm.StartMPI("MPGT parallel segmentation and tracking"); ///Este timer cuenta
             ↪ todo en rid=1, los otros en master
102      tracking_cronometer.CrnStartMPI("Tracking"); ///Este timer cuenta el
             ↪ tracking
103      tracking_cronometer.CrnPauseMPI();
104      //com1_cronometer.CrnStartMPI("Comunications within the tracking");
105      //com1_cronometer.CrnPauseMPI();
106   }
107
108   if(rid==MASTER) {
109      seg_cronometer.CrnStartMPI("Segmentation"); ///Este timer cuenta el
             ↪ segmentation
110      seg_cronometer.CrnPauseMPI();
111      com2_cronometer.CrnStartMPI("Communication between even and odds");
112      com2_cronometer.CrnPauseMPI();
113   }
114
115   if(create_video && (rid==MASTER+1)){
116      writing_cronometer.CrnStartMPI("Wrinting"); ///Este timer cuenta la
             ↪ escritura de video
117      writing_cronometer.CrnPauseMPI();
118   }
119
120   YamlParser yp;
121   yp.LoadConfig(config_file);
122   yp.segparms["video_height"]=dHeight;
123   yp.segparms["video_width"]=dWidth;
124
125   if(rid==MASTER+1){
126      yp.InitResultWriter(track_file);
127      yp.StoreHeaderInfo(input_file, fh.bufVideoInfo);
```

```
128     }
129
130     if(load_init_data)
131        init_tpoints = yp.LoadTrackingFile(init_file, "non");
132
133     ///Segmentator object and operation mode;
134     Segmentator sg(yp.segparms);
135
136     ///Run on each frame on a batch mode equal to the graph window until stop frame
            ↪  % window size to avoid seg faults
137
138     ///int stop_frame_wsize = (int)yp.stop_frame - ((int)yp.stop_frame % yp.
            ↪ graph_window);
139     int total_range_frames = yp.stop_frame - yp.start_frame+1;
140
141     if(nprocs>2){
142        if(((total_range_frames)%((nprocs/2)*yp.graph_window)) != 0){
143           cout << "Warning: Total number of frames must be a multiple of processes
                  ↪ * window size... truncating range" << endl;
144           //if(total_range_frames<((nprocs/2)*yp.graph_window)){
145              //Aca hago el calculo de cuantos realmente se puede procesar
146              int division_tf_wxp = (total_range_frames/((nprocs/2)*yp.graph_window)
                     ↪ );
147              if(division_tf_wxp==0){
148                 cout << "The number of frames must be greater than number of
                        ↪ processes * window" << endl;
149                 com.Stop();
150                 boost_env.~environment();
151                 exit(1);
152              }
153              else{
154
155                 ///Update range and last frame
```

```
156                 total_range_frames = (int)division_tf_wxp * (nprocs/2)*yp.
                        ↪ graph_window-1;//aca puede que vaya un +1
157                 yp.stop_frame = yp.start_frame+total_range_frames;
158                 cout << "New range goes from " << yp.start_frame << " to " << yp.
                        ↪ stop_frame << endl;
159             }
160         }
161     //}
162 }
163 else{
164     if(((total_range_frames)%((nprocs)*yp.graph_window)) != 0){
165         cout << "Warning: Total number of frames must be a multiple of processes
                ↪ * window size... truncating range" << endl;
166         //Aca hago el calculo de cuantos realmente se puede procesar
167         int division_tf_wxp = (total_range_frames/((nprocs)*yp.graph_window));
168         if(division_tf_wxp==0){
169             cout << "The number of frames must be greater than number of
                    ↪ processes * window" << endl;
170             com.Stop();
171             boost_env.~environment();
172             exit(1);
173         }
174         else{
175
176             ///Update range and last frame
177             total_range_frames = (int)division_tf_wxp * (nprocs)*yp.
                    ↪ graph_window-1;//aca puede que vaya un +1
178             yp.stop_frame = yp.start_frame+total_range_frames;
179             cout << "New range goes from " << yp.start_frame << " to " << yp.
                    ↪ stop_frame << endl;
180         }
181     }
182 }
183
```

```
184    int max_iteraciones = total_range_frames/((nprocs/2)*yp.graph_window);
185    cout << "maximo iteraciones = " << max_iteraciones << " total_range_frames = "
           ↪ << total_range_frames
186        << " nprocs = " << nprocs << " window = " << yp.graph_window << " from rid
               ↪ =" << rid << endl;
187
188    Graph bpg_previous_graph,bpg_last_graph;
189    OclusionMemoir OMemory;
190    AbductionMemoir AMemory;
191    OMemory.InitMemoir(yp.mpgtparms,yp.weights);
192    AMemory.InitMemoir(yp.mpgtparms,yp.weights);
193    for(int iteracion=0; iteracion <= max_iteraciones; iteracion++){
194       if(create_video && ((rid%2)==1))
195          fh.StartParallelIterationalVideoSaver(output_file,iteracion);
196       int start_frame_mpi;
197       int stop_frame_mpi;
198       if(nprocs != 2){
199          start_frame_mpi = yp.graph_window * (rid/2) + yp.graph_window * (nprocs
               ↪ /2) * iteracion + yp.start_frame;
200          stop_frame_mpi = start_frame_mpi + yp.graph_window -1;
201       }
202       else{
203          start_frame_mpi = yp.graph_window * iteracion + yp.start_frame;
204          stop_frame_mpi = start_frame_mpi + yp.graph_window -1;
205       }
206
207       ///for(int step=start_frame_mpi;step<isize && step<= stop_frame_mpi;step=
               ↪ step+yp.graph_window){
208
209       MultiPartiteGraph mpgo(yp.graph_window);
210       mpgo.initial_frame=start_frame_mpi;
211       mpgo.last_frame=stop_frame_mpi;
212       D(cout << "Working on block from " << mpgo.initial_frame << " to " << mpgo.
               ↪ last_frame << " for process " << rid << " on iteration " << iteracion
```

```
        ↪  << endl;)
213     D(cout << "Start mpi frame = " << start_frame_mpi << " Stop mpi frame = " <<
        ↪  stop_frame_mpi << " for process " << rid << " on iteration " <<
        ↪ iteracion << endl;)
214
215     ///*************** First step: Segmentation by even processes
        ↪ ***************
216     if((rid%2)==0){
217        for(int i=0; i<yp.graph_window;i++) {
218
219           if(rid==MASTER)
220              seg_cronometer.CrnResumeMPI();
221
222           int frame_number = i+start_frame_mpi;
223           D(cout << "Extracting blobs from frame " << frame_number << endl;)
224           iframe = fh.GetFrame(frame_number);
225           if(iframe.empty())
226              break;
227
228           //cv::normalize( iframe, iframe, 0, 1, cv::NORM_MINMAX, CV_32FC3, cv::
                ↪ Mat() );
229           vector< vector< cv::Point > > contornos_jugadores = sg.Segment(iframe)
                ↪ ;
230
231           ///Adquirir los descriptores para cada blob y guardarlos en los nodos
                ↪ correspondientes
232           for(unsigned int j=0;j<contornos_jugadores.size();j++){
233              PlayerBlob blob;
234              blob.FillModel(frame_number,j,contornos_jugadores[j],iframe);
235              if(load_init_data)
236                 blob.AssignInitTag(init_tpoints,yp.init_tkp_threshold);
237              if(draw_centers==true)
238                 centers.push_back(blob.centroid);
239
```

```
240             ///Quitar los blobs que estan fuera de los limites superiores e
                    ↪ inferiores
241             if(yp.segparms["enable_boundary_blob_remover"]){
242                double high_limit = (double)dHeight*(double)yp.segparms["
                       ↪ lower_boundary"]/100.0;
243                double low_limit = (double)dHeight*(double)yp.segparms["
                       ↪ upper_boundary"]/100.0;
244                //cout << "Blob with id " << blob.blob_id << " on frame " <<
                       ↪ frame_number << " at [ " << blob.centroid.x << " , " << blob
                       ↪ .centroid.y
245                // << " ] inspected" << " lower limit = " <<low_limit << " higher
                       ↪ limit " << high_limit << endl;
246                if((blob.centroid.y <= low_limit) &&
247                   (blob.centroid.y >= high_limit) ){
248                   mpgo.InsertNode(frame_number,blob);
249                }
250             }
251
252             else
253                   mpgo.InsertNode(frame_number,blob);
254                //cout << "Mostrando info de blob " << blob.blob_id << " en el
                       ↪ frame " << blob.blob_frame_number << endl;
255                //imshow("Contorno de jugador",blob.contourRegion);
256             }
257             if(rid==MASTER)
258                seg_cronometer.CrnPauseMPI();
259          }//for loop within a multipartite graph
260
261          if(rid==MASTER)
262             com2_cronometer.CrnResumeMPI();
263          com.SendMultiGraph(mpgo.mpg,rid,rid+1);
264          if(rid==MASTER)
265             com2_cronometer.CrnPauseMPI();
266
```

```
267       }//if are a even proc
268
269       ///*************** Second step: Tracking by odd processes***************
270       if(rid%2==1){
271          if(enable_tracking){
272
273             D(cout<<"Receiving a multigraph " << " source " << rid-1 << " dest "
                      ↪ << rid << " on iteration " << iteracion << endl;)
274 //    com_cronometer.CrnResumeMPI();
275             mpgo.mpg=com.ReceiveMultiGraph(rid-1,rid,yp.graph_window);
276 //    com_cronometer.CrnPauseMPI();
277
278             if(rid==(MASTER+1))
279                tracking_cronometer.CrnResumeMPI();
280
281             ///Caso general de pares
282             if(nprocs!=2){
283                if((start_frame_mpi != yp.start_frame) && (iteracion!=(
                         ↪ max_iteraciones))){///To exclude first frame of sequence.
284
285                   if(rid==1)
286                      bpg_previous_graph=com.ReceiveGraph(nprocs-1,1);
287                   else
288                      bpg_previous_graph=com.ReceiveGraph(rid-2,rid);
289
290                   //mpgo.SolveBorderBPG(yp.weights,step,bpg_previous_graph);
291                   mpgo.SetOclusionMemories(OMemory.vector_memoir);
292                   mpgo.SetAbductionMemories(AMemory.vector_memoir);
293                   mpgo.SolveExtendedBorderBPG(yp.weights,yp.mpgtparms,
                            ↪ start_frame_mpi,bpg_previous_graph);
294                   bpg_previous_graph.clear();
295                   OMemory.vector_memoir.clear();
296                   AMemory.vector_memoir.clear();
297                }
```

```
298
299             else if(iteracion==max_iteraciones){
300
301                 if(rid==1)
302                     bpg_previous_graph=com.ReceiveGraph(nprocs-1,1);
303                 else
304                     bpg_previous_graph=com.ReceiveGraph(rid-2,rid);
305
306                 //mpgo.SolveBorderBPG(yp.weights,step,bpg_previous_graph);
307                 mpgo.SetOclusionMemories(OMemory.vector_memoir);
308                 mpgo.SetAbductionMemories(AMemory.vector_memoir);
309                 mpgo.SolveExtendedBorderBPG(yp.weights,yp.mpgtparms,
                        ↪ start_frame_mpi,bpg_previous_graph);
310                 bpg_previous_graph.clear();
311                 ///OMemory.vector_memoir.clear();
312                 ///AMemory.vector_memoir.clear();
313             }
314
315             //cout << "step = " << step << endl;
316             //mpgo.MVSimpleSolver(yp.weights,yp.mpgtparms,step);
317             //mpgo.MVMPSolver(yp.weights,step);
318             mpgo.MVMPAssignFromFirstGraph(yp.weights,start_frame_mpi);
319             mpgo.MVExtendedSimpleSolver(yp.weights,yp.mpgtparms,start_frame_mpi
                    ↪ );
320
321             int last_frame_number_from_window = stop_frame_mpi;
322             //cout << "bpg first graph = " << last_frame_number_from_window <<
                    ↪ endl;
323             bpg_last_graph = mpgo.GetGraph(last_frame_number_from_window);
324             OMemory.vector_memoir = mpgo.GetOclusionMemories();
325             AMemory.vector_memoir = mpgo.GetAbductionMemories();
326
327             ///for(int gi=0; gi<yp.graph_window;gi++) {
328             /// int frame_number = gi+step;
```

```
329                /// yp.StoreFrameResult(mpgo.GetGraph(frame_number));
330                ///}
331
332                if(start_frame_mpi == yp.start_frame){
333                   if(rid<nprocs-1)
334                      com.SendGraph(bpg_last_graph,rid,rid+2);
335                      bpg_last_graph.clear();
336                }
337
338                if((start_frame_mpi != yp.start_frame) && (iteracion!=(
                       ↪ max_iteraciones))){
339                   if(rid<nprocs-1){
340                      com.SendGraph(bpg_last_graph,rid,rid+2);
341                      bpg_last_graph.clear();
342                   }
343                   else {
344                      com.SendGraph(bpg_last_graph,rid,1);
345                      bpg_last_graph.clear();
346                   }
347                }
348                else if(iteracion==max_iteraciones){
349                   if(rid<nprocs-1){
350                      com.SendGraph(bpg_last_graph,rid,rid+2);
351                      bpg_last_graph.clear();
352                   }
353                }
354             }
355
356             ///Caso particular cuando son solo 2 procesos
357             else{
358                if((start_frame_mpi != yp.start_frame) && (iteracion!=(
                       ↪ max_iteraciones))){///To exclude first frame of sequence.
359                D(cout << "El tamano del grafo previo es " << bpg_last_graph.
                       ↪ size() << " estoy en iteracion " << iteracion << endl;)
```

```
360                 mpgo.SetOclusionMemories(OMemory.vector_memoir);
361                 mpgo.SetAbductionMemories(AMemory.vector_memoir);
362                 mpgo.SolveExtendedBorderBPG(yp.weights,yp.mpgtparms,
                        ↪ start_frame_mpi,bpg_last_graph);
363                 OMemory.vector_memoir.clear();
364                 AMemory.vector_memoir.clear();
365
366                 ///mpgo.MVMPAssignFromFirstGraph(yp.weights,start_frame_mpi);
367                 mpgo.MVExtendedSimpleSolver(yp.weights,yp.mpgtparms,
                        ↪ start_frame_mpi);
368                 bpg_last_graph=mpgo.GetGraph(stop_frame_mpi);
369                 OMemory.vector_memoir = mpgo.GetOclusionMemories();
370                 AMemory.vector_memoir = mpgo.GetAbductionMemories();
371             }
372             else if(iteracion==(max_iteraciones)){
373                 D(cout << "El tamano del grafo previo es " << bpg_last_graph.
                        ↪ size() << " estoy en iteracion " << iteracion << endl;)
374                 mpgo.SetOclusionMemories(OMemory.vector_memoir);
375                 mpgo.SetAbductionMemories(AMemory.vector_memoir);
376                 mpgo.SolveExtendedBorderBPG(yp.weights,yp.mpgtparms,
                        ↪ start_frame_mpi,bpg_last_graph);
377                 ///OMemory.vector_memoir.clear();
378                 ///AMemory.vector_memoir.clear();
379
380                 mpgo.MVExtendedSimpleSolver(yp.weights,yp.mpgtparms,
                        ↪ start_frame_mpi);
381                 OMemory.vector_memoir = mpgo.GetOclusionMemories();
382                 AMemory.vector_memoir = mpgo.GetAbductionMemories();
383             }
384
385             else{
386                 bpg_last_graph=mpgo.GetGraph(stop_frame_mpi);
387             }
388         }
```

```
389             if(rid==MASTER+1)
390                tracking_cronometer.CrnPauseMPI();
391          }
392          else{
393             cout << "No tracking step selected!!!!!!!" << endl;
394          }
395       }//if odd rid
396
397       ///*************** Third step: Draw results ***************
398       if(create_video){
399          com.Barrier();
400          if((rid%2)==1){
401             if(rid==MASTER+1)
402                writing_cronometer.CrnResumeMPI();
403
404             ///Dibujar los contornos y los ids para cada frame del multigrafo
405             for(int gi=0; gi<yp.graph_window;gi++) {
406
407                int frame_number = gi+start_frame_mpi;
408                cout << "Drawing results for frame " << frame_number << endl;
409                iframe = fh.GetFrame(frame_number);
410                if(iframe.empty())
411                   break;
412
413                vector< vector< cv::Point > > contornos_jugadores;
414                vector< Node > vectornode;
415
416                Graph solved_graph = mpgo.GetGraph(frame_number);
417
418                ///Extract the contours from the graph nodes
419                for(unsigned int ni=0;ni<solved_graph.size();ni++){ //ni node id
420                   Node blobnode = mpgo.GetNode(frame_number,ni);
421                   contornos_jugadores.push_back(blobnode.contorno);
422                   vectornode.push_back(blobnode);
```

```
423                 }//for loop to sweep all nodes within a graph
424
425                 cv::Mat contornos_graf = DrawContour( contornos_jugadores, iframe.
                        ↪ rows, iframe.cols );
426                 if(draw_centers==true){
427                    for(unsigned int j=0;j<centers.size();j++)
428                       circle( contornos_graf, centers[j], 3, cv::Scalar(255), -1,
                              ↪ 8, 0 );
429                 }
430                 contornos_graf = ConvertColor( contornos_graf, PRIS_GRY2RGB );
431                 cv::normalize( iframe, iframe, 0, 1, cv::NORM_MINMAX, CV_32FC3, cv
                        ↪ ::Mat() );
432                 cv::Mat oframe = Add( iframe, contornos_graf );
433                 oframe = Normalize( oframe, PRIS_8UC1 );
434                 if(oframe.empty()){
435                    cout << "Frame de salida vacio" << endl;
436                    break;
437                 }
438                 ///Dibujar los blob ids/labels
439                 if(draw_ids==true){
440                    for(unsigned int vn = 0; vn < vectornode.size(); vn++){
441                       D(cout << "Drawing id " << vectornode[vn].player_tag << " for
                              ↪  frame " << frame_number << " from rid " << rid <<
                              ↪ endl;)
442                       std::string sid = std::to_string(vectornode[vn].player_tag);
443                       putText(oframe, sid, vectornode[vn].centroid,
                              ↪ FONT_HERSHEY_PLAIN, 1.0, CV_RGB(0,255,0), 2.0);
444                    }
445                 }
446
447                 //imshow("Video de salida",oframe); waitKey();
448                 fh.SimpleFrameSaver(oframe);
449                 yp.StoreFrameResult(mpgo.GetGraph(frame_number)); //Testing stuff
450              }
```

```
451             if(rid==MASTER+1)
452                writing_cronometer.CrnPauseMPI();
453          }//ridimpar
454       }//for loop to sweep all frames in a batch mode
455    ///}
456    }///Iteracion
457
458    com.Barrier();
459    if(rid==MASTER){
460       seg_cronometer.CrnStopMPI();
461       com2_cronometer.CrnStopMPI();
462    }
463    if(rid==MASTER+1){
464       tm.StopMPI();
465       //com_cronometer.CrnStopMPI();
466       tracking_cronometer.CrnStopMPI();
467    }
468    if(create_video && (rid==MASTER+1))
469       writing_cronometer.CrnStopMPI();
470    if(rid==MASTER+1){
471       cout << "FPS = " << (yp.stop_frame-yp.start_frame)/tm.accum << endl;
472       yp.CloseResultWriter();
473    }
474    cout << "Process " << rid << " about to end" << endl;
475    com.Stop();
476
477    return 0;
478
479 }
480
481 ///**********************Funcion que imprime ayuda
482 void PrintHelp(void)
483 {
484    std::cout <<
```

```
485          "\n Arguments: \n"
486          "-i <ifile>: Input video file\n"
487          "-o <ofile>: Output video file\n"
488          "-t <tfile>: Track file to store tracking information\n"
489          "-c <cfile>: Configuration file\n"
490          "-l <lfile>: Initialization data file\n"
491          "-C: Draw blob trayectories \n"
492          "-I: Draw blob ids \n"
493          "--help: Show help\n";
494
495    exit(1);
496 }
497
498 ///**********************Funcion que procesa argumentos
499 void ProcessArgs(int argc, char** argv)
500 {
501
502    //if (argc < 4)
503    // PrintHelp();
504
505    const char* const short_opts = "i:o:t:c:l:CIh";
506    const option long_opts[] = {
507          {"input", required_argument, nullptr, 'i'},
508          {"output", optional_argument, nullptr, 'o'},
509          {"track", optional_argument, nullptr, 't'},
510          {"config", optional_argument, nullptr, 'c'},
511          {"load", optional_argument, nullptr, 'l'},
512          {"centers", no_argument, nullptr, 'C'},
513          {"ids", no_argument, nullptr, 'I'},
514          {"help", no_argument, nullptr, 'h'},
515          {nullptr, 0, nullptr, 0}
516    };
517
518    while (true)
```

```
519    {
520       const auto opt = getopt_long(argc, argv, short_opts, long_opts, nullptr);
521
522       if (-1 == opt)
523          break;
524
525       switch (opt)
526       {
527       case 'i':
528          input_file = std::string(optarg);
529          if(rid==MASTER)
530             std::cout << "Input video file is: " << input_file << std::endl;
531          break;
532
533
534       case 'o':
535          output_file = std::string(optarg);
536          if(rid==MASTER)
537             std::cout << "Output video file set to: " << output_file << std::endl;
538          create_video=true;
539          break;
540
541       case 't':
542          track_file = std::string(optarg);
543          if(rid==MASTER)
544             std::cout << "Tracking file set to: " << track_file << std::endl;
545          enable_tracking=true;
546          break;
547
548       case 'c':
549          config_file = std::string(optarg);
550          if(rid==MASTER)
551             std::cout << "Configuration file set to: " << config_file << std::endl
                  ↪ ;
```

```
552            break;
553
554         case 'l':
555            init_file = std::string(optarg);
556            if(rid==MASTER)
557               std::cout << "Initialization file set to: " << init_file << std::endl;
558            load_init_data = true;
559            break;
560
561         case 'C':
562            draw_centers = true;
563            if(rid==MASTER)
564               std::cout << "Drawing segmented trayectories " << std::endl;
565            break;
566
567         case 'I':
568            draw_ids = true;
569            if(rid==MASTER)
570               std::cout << "Drawing blob ids " << std::endl;
571            break;
572
573         case 'h': // -h or --help
574         case '?': // Unrecognized option
575         default:
576            if(rid==MASTER)
577               PrintHelp();
578            break;
579         }
580
581      }
582
583   }
```