

<https://doi.org/10.1007/s11042-013-1810-4>

Access to this work was provided by the University of Maryland, Baltimore County (UMBC) ScholarWorks@UMBC digital repository on the Maryland Shared Open Access (MD-SOAR) platform.

Please provide feedback

Please support the ScholarWorks@UMBC repository by emailing scholarworks-group@umbc.edu and telling us what having access to this work means to you and why it's important to you. Thank you.

MultiCAMBA: A System for Selecting Camera Views in Live Broadcasting of Sport Events Using a Dynamic 3D Model

Roberto Yus^a, Eduardo Mena^a, Sergio Ilarri^a, Arantza Illarramendi^b, Jorge Bernad^a

^a*University of Zaragoza, María de Luna 1, Zaragoza, Spain*

^b*Basque Country University, Manuel de Lardizábal s/n, San Sebastián, Spain*

Abstract

For a Technical Director (TD) in charge of a live broadcasting, selecting the best camera shots among the available video sources is a challenging task, even more now that the number of cameras (some of them mobile, or attached to moving objects) in the broadcasting of sport events is increasing. So, the TD needs to manage a great amount of continuously changing information to quickly select the camera whose view should be broadcasted. Besides, the better the decisions made by the TD, the more interesting the content for the audience. Therefore, the development of systems that could help the TD with the selection of camera views is demanded by broadcasting organizations.

In this paper, we present the system MultiCAMBA that helps TDs in the live broadcasting task by allowing them to indicate in run-time their interest in certain kind of shots, and the system will show the cameras that are able to provide them. To achieve this task, the system manages location-dependent queries generated according to the interests of the TD. Moreover, to avoid the use of costly on line real-image processing techniques over the camera views, such real camera views are recreated in a 3D engine by using the information contained in a 3D model of the scenario. This model is updated continuously with real-time data retrieved from the real objects and cameras in the scenario. In this way, the system extracts high-level semantic features

Email addresses: ryus@unizar.es (Roberto Yus), emena@unizar.es (Eduardo Mena), silarri@unizar.es (Sergio Ilarri), a.illarramendi@ehu.es (Arantza Illarramendi), jbernad@unizar.es (Jorge Bernad)

of 2D projections of the 3D reconstruction of the camera views. We present a prototype of the system and experimental results that show the feasibility of our proposal.

Keywords: Content selection in run-time, mobile multi-camera management, location-aware systems

1. Introduction

There exist many scenarios where it is important to select among many cameras the one whose view is the most interesting. For example, in the live broadcasting of sport events, a Technical Director (TD) has to make quick decisions to select the camera video stream to broadcast. Furthermore, nowadays broadcasting organizations are increasing the number of cameras covering sport events (e.g., *Sky TV* uses 24 cameras in Premier League matches). The higher the number of cameras available, the richer the content that can be obtained, and therefore more complicated is for the TD to select the best one. Therefore, it would be interesting to develop systems that could help the TD to select the most interesting camera view among many sources.

For example, consider the TD in charge of the live broadcasting of a rowing race (this is a very popular sport along the north of Spain). From the broadcasting perspective, the technology and equipment involved in the event nowadays include multiple cameras (in sailing boats, in a helicopter, in the harbor, in a nearby island, etc.) and a GPS receiver on every boat. In this context, it would be very helpful to have a system where the TD could define his/her interest on a certain view (e.g., *a view of the front of two rowing boats*) and obtain the list of cameras that could provide it.

A relevant feature of such a system would be to support the possibility of analyzing camera views in real-time to extract enough information to find out what they are showing. For this task, a popular approach is to process the real images provided by the cameras. However, using real image processing techniques to extract *high-level features* related to the semantics of the scene, such as the kind of objects or the specific identity of the object, is a challenge (and even more in real-time). This is related to the problem of the well known “semantic gap” that exists between low-level features and high-level semantics, which has attracted considerable research attention (e.g., see [1] and [2]). For example, consider the camera shot in Fig. 1(a), where all the rowing boats are shown from a large distance to allow the viewer to have a

general overview of the race. Real image processing techniques would face two main problems:



Figure 1: Real camera footage (a) and interesting and other objects in the scene (b).

1. Along with the rowing boats there exist multiple moving objects (judges, support team, etc.) very close to them (in Fig. 1(b) we have highlighted the rowing boats and the other moving objects with red dotted and yellow circles, respectively). So, it will be difficult to distinguish the *objects-of-interest* (rowing boats) from the other objects in the scenario based on their visual features and moving patterns.
2. Even if the objects that are rowing boats could be identified, the TD could be interested in a specific boat (e.g., “Kaiku” in Fig. 1(b), highlighted with an arrow). Identifying this boat automatically among the others will be very difficult.

In this paper, to overcome these difficulties, we present the system *MultiCAMBA* (*Multi-CAMera Broadcasting Assistant*) that uses a different approach. Instead of on line analyzing the real images provided by the cameras, such real camera views are recreated in a 3D engine by using the information contained in a 3D model of the scenario. The system manages and keeps this 3D model up-to-date in real-time according to the information of objects-of-interest (identification, location, direction, approximate extent, etc.) and cameras (location, direction, Field of View –FOV–, etc.) in the scenario. In this way, the system obtains projections of the 3D reconstruction of a camera view to extract high-level semantic features of the real camera view. So, the proposed system is able to automatically and accurately detect the specific objects that are viewed by a camera (e.g., “the Kaiku rowing boat” vs. simply “a boat”). Moreover, the system obtains other high-level features

of each object detected in a camera view, such as: their amount visible in the camera view (as a percentage), the viewpoint of the camera concerning each object (e.g., it could view the front and top of the object), the amount of the shot occupied by them (that determines the space available for uninteresting objects), etc. We present in this paper the efficient methods that we have developed to obtain this information continuously and in real-time (in our tests, every second), which makes the system suitable for live broadcasting.

Due to the dynamic nature of sport event scenarios (objects and cameras can move and camera views can change), the system must reevaluate continuously the *location-dependent queries* [3] posed by the user¹ (e.g., *obtain cameras closer than 40 meters from a rowing boat*). These queries are generated based on the requirements of the TD expressed through a user-friendly Graphical User Interface (GUI). The TD can indicate in the GUI the specific view he/she is interested in, and the results obtained by the system (the cameras that could satisfy the requirements of the TD) will be presented in a 3D reconstruction of the scenario.

In summary, the main contribution of our proposal is the development of a system that:

- Enables the TD to indicate in run-time his/her interest in certain kinds of shots through a user-friendly GUI, and the system selects (from the available sources) the cameras that can provide such interesting shots.
- Processes in real-time the views provided by the cameras without the need to analyze real images. For this purpose, it makes use of a 3D model updated continuously with real-time data from the scenario.
- Obtains high-level semantic features of the camera views efficiently enough for real-time processing, using a 3D engine and the up-to-date information of the 3D model.

So, as long as the locations of the objects-of-interest and cameras (and an approximation of the extent of the objects) can be obtained in real-time, the system presented can be applied to any context, as no assumption is made regarding the number of cameras in the scenario (the view of each one

¹Notice that the Technical Director is the user of the system and we will use both terms to refer to him/her.

can be analyzed separately), the kind of scenario (the system can be used in scenarios involving moving objects, cameras, and queries about them), and the positioning mechanism used to obtain the locations of the objects and cameras. In some situations it could be challenging to obtain this information for certain objects (e.g., it could be difficult to obtain the real-time precise location of a ball or the extent of soccer players that move their limbs while running). However, our approach does not rely on a specific technology to obtain this information nor requires 100% precision of these data to effectively distinguish between cameras that are interesting or not for a given query.

The rest of this paper is structured as follows. We review some related works in Section 2. Then, we present the architecture of the system in Section 3. We describe the approach proposed to analyze the views of the cameras using the 3D model of the scenario in Section 4. We explain the GUI proposed for the TD in Section 5; the GUI allows the TD to easily define his/her queries and displays the results obtained. We then present some experiments performed to validate our proposal in Section 6. Finally, conclusions and future work are included in Section 7.

2. Related Work

Up to the authors' knowledge, no other work has focused on the real-time selection of camera views based on the extraction of high-level features of images provided by multiple moving cameras using a 3D model of the scenario. However, there is extensive research on the analysis of real images to obtain what a camera is viewing. These studies can be classified according to the kind of processing performed on the video streams (on-line or offline). Proposals to process the videos on-line usually take into account the *cinematic* features of the views provided by the cameras, such as the shot types (e.g., a long shot, a close-up shot, etc.). Considering high-level features related to the semantics of the scene, such as the specific object, the visible amount of the object, or *object-based* features, such as the color and shape of the objects, their interactions, etc., may be computationally too costly for on-line processing, even though it would provide richer semantic details. Works to analyze videos offline, and so having more time for the processing, use object-based features to extract events of interest (e.g., a pitching scene in a baseball video) and usually consider only static cameras.

2.1. Real-Time Camera Selection for Sport Events

We can mention [4, 5], that share the goal of our proposal of selecting camera views in real-time for TV broadcasting, even though their approaches are based on analyzing the real images provided by the cameras.

The context of the system presented in [4] is soccer games. It relies on the well-defined structure of a soccer broadcast to alternate the selection between cameras that provide a far view and cameras that provide a medium/close-up view. The view switching method proposed in that paper does not analyze high-level features of the camera views and the authors assume that all the cameras are following the game action (and hence they have a similar content). So, their problem is to select those cameras providing a clear view (they discard blurry images). The main differences between our work and [4] is that we allow the TD to define the criteria to be considered to select a camera view, and moreover our proposal is able to obtain a good number of high-level features of a camera view in live. Besides, we make no assumptions about the current views of the cameras.

In [5] a system is also presented to automatically select in live the camera to broadcast in a soccer game. As we do, the authors also consider low-cost cameras as a way to reduce the costs of a sport broadcasting. They assume that there exist four cameras located along the field and they process their views to obtain the size of the ball in each one. With this information, they propose to select the camera whose view shows the largest area of the projected ball. Therefore, their approach is focused on ball sports and under the assumption that the best views are those that provide a better view of the ball, while ours can be applied to other contexts where selecting among many camera views is needed.

The goal of these two works is to select automatically the best camera to broadcast in soccer events based on parameters as image quality. However, our approach, that is not focused in any specific sport, enables the TD to define the kind of camera he/she wants to broadcast based on the objects that this camera views.

2.2. Assistants for Sport Videos Summarization

A number of works have focused their efforts on the specific problem of helping producers of sport videos. For example, as part of the *APIDIS* project, in [6] a system that helps the video production and video summarization in the context of basketball games is presented. The work presented in [7] tackles the problem of summarizing videos of soccer games by applying

different image processing algorithms to analyze the input videos extracting cinematic and object-based features. In [8] the problem of video summarization, based on metadata describing the semantic content of MPEG-7 videos, is considered in the context of baseball games. Cricket videos, as well as soccer videos, are used to validate the work in [9], which exploits audio features (such as an increase in the audio level of the voice of the commentators or the cheers of the audience) to extract excitement clips from sport videos. Along the same line, the work in [10] benefits from audio and motion cues to extract highlights from baseball videos. Textual overlays appearing in images are exploited in [11] to create personalized summaries of American football videos (i.e., video abstracts that take into account the user preferences); similarly, works such as [12, 13] use webcast text associated to the video for event detection. In [14] the authors focus on the problem of ranking, structuring, and summarizing highlights to match a user’s personalized query, within the context of racket games (tennis and badminton). Like [14], most works in this area emphasize the importance of taking into account the user preferences and/or expectations [15]. A detailed survey of soccer video analysis systems can be found at [16].

All these works are concerned about facilitating the production of sport events, like the proposal in this paper, but they have a different purpose. Thus, the purpose of these works is usually to perform an automatic video production in an offline setting (so, for example, achieving a good performance for real-time processing is not an issue) by using real image processing techniques to extract low-level features (e.g., color, texture, shapes, etc.) that will be processed to obtain cinematic features (i.e., shot classification).

2.3. Camera Management for Broadcasting

Several works in the literature have considered the problem of automatic camera management for recording and broadcasting lectures and talks. For example, in *AutoAuditorium* [17] two cameras and microphones are used to obtain information about what is happening on the stage and perform an automatic audio mixing, tracking of the people on stage, and camera selection. Another interesting work is [18], which implements several production rules, inspired by the way professional video producers work, in order to take the appropriate recording decisions. The system *FlySPEC* [19] combines a PTZ (Pan-Tilt-Zoom) camera and a panoramic camera and benefits from the involvement of the audience, participating through explicit requests, to reduce the probability of unsatisfactory recordings. The *Microsoft Research*

LecCasting System (MSRLCS) [20] supports a scripting language to facilitate the customization of production rules for different room configurations and production styles. As a final example, the *Virtual Videography* [21] advocates an offline processing to have more time and information to perform the video production.

Although the context and purpose of these works is different from the ones considered in this paper, they highlight the interest of the development of automatic video production techniques to save production costs and enable fast access to multimedia information. Several other works focus on multi-camera management (e.g., [22, 23]). However, they usually consider only cameras that are static (i.e., at fixed locations), whereas the cameras considered in our proposal can move.

3. Overview of the System

In this section we provide an overview of the steps followed by the system to accomplish the goal of obtaining the camera views that fulfill the requirements of the TD (see Fig. 2):

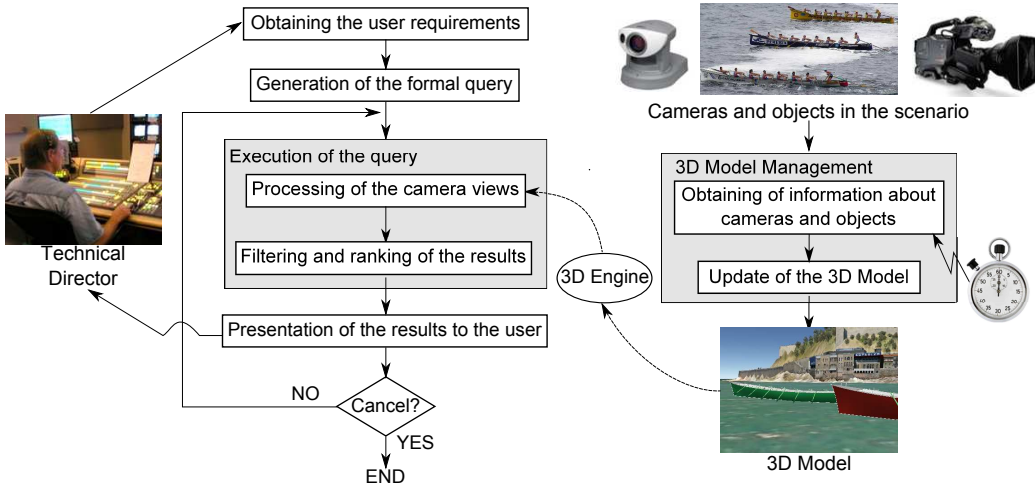


Figure 2: Main steps followed by the system.

1. *3D model management* (see Section 3.1). The 3D model of the scenario is an essential part of our system as it stores the information about the different objects and cameras in the scenario. The system maintains this 3D model updated in parallel to the processing of user queries.

2. *Obtaining the user requirements.* The requirements of the user are captured through an easy-to-use interface (see Section 5). The system provides two mechanisms for that: a) the TD selects from lists the target object/s that must be part of the view and the constraints that the camera view must fulfill, or b) the TD clicks on predefined queries. Complementary, the ideas presented in [24] can be applied to enable the TD to define his/her requirements through an interface for the definition of 3D scenes. In addition, touchable 3D interfaces [25] could be helpful to improve the immersion of the TD in this process, although this is out of the scope of this paper.
3. *Generation of the formal query* (Section 3.2). By analyzing the information provided by the user, the system generates a location-dependent query capturing his/her requirements.
4. *Execution of the query* (Section 4). The system obtains high-level features of the camera views (objects viewed, amount of them covered, kind of view, etc.) and the cameras are filtered to obtain those whose view fulfills the user requirements. Then, the answer set is ranked according to the user preferences.
5. *Presentation of the results to the user.* The results obtained by the system are presented to the user in the GUI, both in a tabular form and in a 3D reconstruction of the scenario (see Section 5).

In the following sections we explain with more details these steps.

3.1. 3D Model Management

Obtaining the cameras that are able to provide the TD with the required view is possible thanks to the use of an up-to-date 3D model. The system efficiently keeps the 3D model updated with the information of the objects and cameras in the scenario (obtained from different sensors). This is not an overload for the system (as it only involves obtaining the interesting information and storing it) and it can even be performed in another computer. The 3D model stores the following information about the objects involved in the scenario: identification, location and direction, extent, and front and top vectors (see Fig. 3).

The location and direction of an object, that can be provided, for example, by a GPS and a compass (as in the scenario we have tested in Section 6), have to be continuously updated to obtain accurate results, as they are highly-dynamic data. The imprecision of the localization mechanism could lead

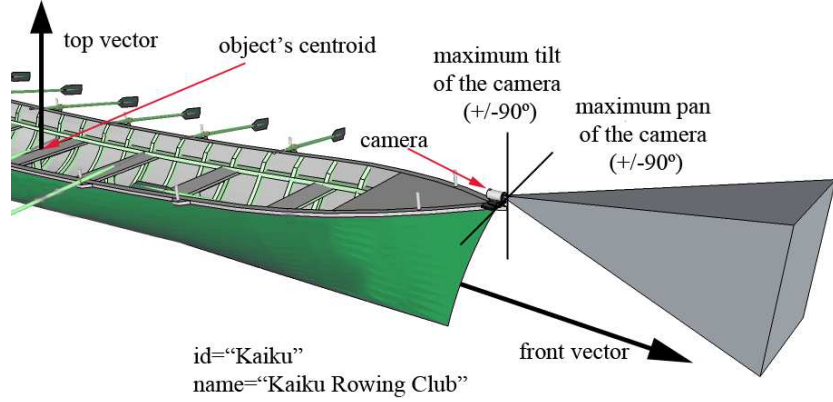


Figure 3: An object-of-interest modeled in our system.

to imprecise answers (e.g., in [26] the authors report an accuracy of around 1 meter for some GPS receivers), but our approach is independent of the specific location mechanism used. So, it is possible to combine, if needed, several positioning mechanisms to increase the accuracy, even for indoor events (by using overhead cameras, sensors, Wi-Fi signal strength maps, etc.).

We want to extract some high-level features, interesting for TDs, of objects inside the FOV of a camera, such as the percentage of them visible or the viewpoint obtained. Therefore, we need to represent these objects in the 3D space using the approximate volume (extent) of space that these objects occupy. However, our system does not need a precise 3D mesh of these object to accomplish its main goal of discarding non-interesting cameras (as we show in our tests in Section 6, where we used an approximate extent for the different types of objects-of-interest). Of course, the more precise the extent of an object-of-interest provided to the system, the more accurate the information it will obtain regarding the percentage of the object viewed by a camera. Thus, users could generate a simple 3D model for these objects or even search for similar already-generated meshes in 3D model databases (e.g., by using keywords or even real images, as studied in [27, 28]).

As the extent of objects-of-interest could change dynamically during the event it could be interesting to provide the system with this information. However, in real life only small parts of these extents change (e.g., the rows of a rowing boat, the limbs of a soccer player, etc.). Thus, the general accuracy of our system will not be affected significantly if non-deformable extents for objects-of-interest are used (in our tests we used fixed 3D models).

Besides, the front and top vectors must be defined for this extent in order to allow the system to distinguish between the different kinds of views of an object (top/bottom, front/rear, left/right, or any combination of two or three elements chosen from the three previous pairs). In this way, the system can answer queries retrieving, for example, cameras recording a top view of an object. In our system, 90-degree angles are considered between the front and top vectors, the sides and the front, and the sides and the top. Thus, no more than three viewpoints are going to be usually selected at the same time in a query for the same object.

Concerning the cameras, which play a key role in the system, we consider that they can rotate (both vertically and horizontally) and change their location (if they are attached to moving objects). We model a camera c as shown in Fig. 4. In the figure, we identify several elements: β_h and β_v are the horizontal and vertical angle of view (that define the *Field of View* –*FOV*– of the camera), respectively; α , α_{max} , α_{min} , and α_{speed} are the current pan, the maximum pan possible, the minimum pan possible, and the pan speed (degrees/second) of such a camera, respectively; finally, θ , θ_{max} , θ_{min} , and θ_{speed} are the current tilt, the maximum tilt possible, the minimum tilt possible, and the tilt speed (degrees/second), respectively². Angles that represent a pan to the right (α) or a tilt upwards (θ) from the corresponding vector are considered positive and those that represent a pan to the left (α) or a tilt downwards (θ) are considered negative. Besides, each camera has a unique identifier.

3.2. Generation of the Formal Query

The system generates queries using the requirements defined by the TD. These queries are expressed using an SQL-like syntax with the following structure:

```

SELECT  projections
FROM    sets-of-objects
WHERE   boolean-conditions
[ ORDER BY  sorting-criteria ]

```

where *projections* is the list of attributes the TD is interested in, *sets-of-objects* is a list of the kinds of objects (e.g., rowing boats, helicopters, etc.)

²In this work we do not deal with the possibility of zooming.

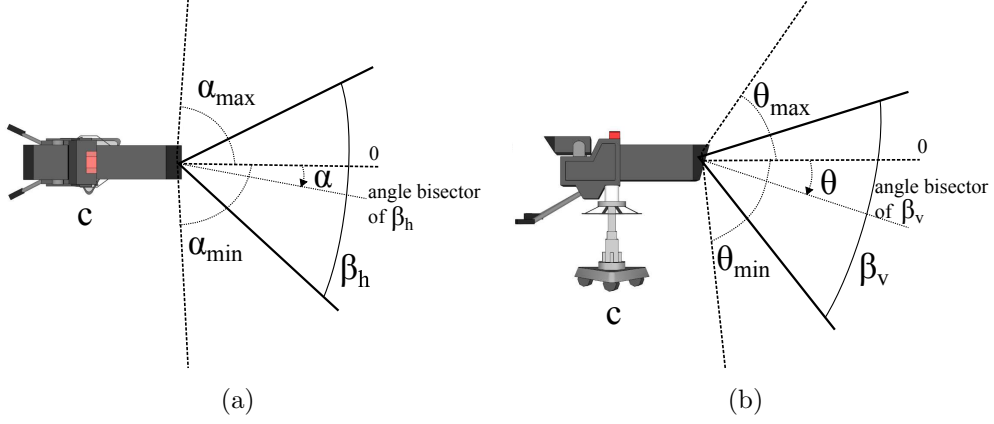


Figure 4: Modeling a camera: pan (horizontal plane) (a), and tilt (vertical plane) (b).

interesting for the query, *boolean-conditions* is a boolean expression (composed by location-dependent constraints [3] and other constraints) that must be true for the objects retrieved by the query, and *sorting-criteria* is the ordering criteria that will be used for the presentation of the results. The *ORDER BY* clause is optional, as in standard SQL. However, the sorting criteria can be particularly interesting when dealing with queries that retrieve cameras, as several cameras may satisfy the query constraints and some criteria is needed to show the most promising results first.

To extend the previous SQL-like syntax in order to support all the features of our proposal, we define the following functions:

- *checkKindOfView(target, cam, <views>)* returns a vector containing a *true* value for each kind of view in the vector *<views>* that the camera *cam* is obtaining of the target object *target*. Another variant of this function is *checkKindOfView(target, cam, <views>, t)*, that performs the same operation but taking into account the estimated view that the camera will obtain after *t* seconds.
- *percentageShot(target, view, cam)* returns the percentage of the shot of the camera *cam* occupied by the target object *target*; *percentageShot(target, view, cam, t)* performs the same operation for the estimated view that the camera will obtain after *t* seconds. Notice that if the user selects a specific viewpoint for the *view* parameter, which is indeed optional, these functions will obtain the amount of the shot occupied by the viewpoint of the target object.

- *percentageObject(target, view, cam)* returns the percentage of the target object *target* that the camera *cam* is viewing; *percentageObject(target, view, cam, t)*, performs the same operation for the estimated view that the camera will obtain after *t* seconds. If the user selects a specific viewpoint for the *view* parameter, which is optional, these functions will obtain the percentage of the viewpoint covered.
- *preferenceDegree(target, cam, α)* returns a numeric value that allows the system to rank cameras that fulfill the user requirements according to how well their views fit his/her preferences. The user sets α , which represents the importance of the percentage of the shot occupied by the target with respect to the percentage of the object viewed (which will have a weight of $1 - \alpha$). In our prototype we advocate computing the preference degree as follows, in order to represent that the higher the percentage the better, but any other function could be used:

$$percentage_of_shot_occupied * \alpha + percentage_of_target_viewed * (1 - \alpha)$$

- *rotationToView(target, cam)* returns a vector with the pan and tilt angles that the camera *cam* has to turn to view the target object *target*. This function makes use of *panToView(target, cam)* and *tiltToView(target, cam)*, that obtain the pan and tilt needed to view the target, respectively.
- *timeToView(cam, <pan, tilt>)* returns the time needed by a camera to turn horizontally *pan* degrees and vertically *tilt* degrees. This function makes use of *timeToPan(cam, pan)* and *timeToTilt(cam, tilt)*, that obtain the time needed to pan and tilt a certain angle, respectively.
- *distance(target, cam)* returns the distance between a camera *cam* and a target object *target*.

To show the use of this SQL-like syntax (a preliminary version can be found at [29]), we first consider an example where the TD asks the system about *cameras that view right now at least 30% of the “Kaiku” boat that fills at least 10% of the shot*. The request will be translated to the following query (we assume that for the TD the percentage viewed is more important than the percentage of the shot occupied and considers $\alpha = 0.4$):

```

SELECT  O.cam.id, score
FROM    Objects AS O
WHERE   <pan,tilt>=rotationToView(Kaiku, O.cam)
        AND pan=0 AND tilt=0
        AND percentageObject(Kaiku, 'any', O.cam) ≥ 0.3
        AND percentageShot(Kaiku, 'any', O.cam) ≥ 0.1
        AND score=preferenceDegree(Kaiku, O.cam, 0.4)
ORDER BY score DESC

```

As the TD wants to obtain the cameras viewing the target object right now, the query includes a condition (*pan=0* and *tilt=0*) that ensures that the cameras in the answer set fulfill this constraint. Besides, the function *preferenceDegree* is used to take into account the TD preferences in the ranking. It is interesting to highlight that the system evaluates the functions *percentageObject*, *percentageShot* and *preferenceDegree* at the same time, using a single rendering of the view and with a single pass (see Section 4.1).

Now, considering that the TD asks about *cameras that can view the front, top and side of “Kaiku” in less than 20 seconds, sorted by the percentage of the target viewed and the time needed to view it* (the largest the percentage and the shorter the time, the more appropriate a camera is), the system generates the following query:

```

SELECT  O.cam.id, pct, time, pan, tilt
FROM    Objects AS O
WHERE   <pan,tilt>=rotationToView(Kaiku, O.cam)
        AND time=timeToView(O.cam, <pan,tilt>)
        AND time < 20
        AND views=checkKindOfView(Kaiku, O.cam,
        <front, top, side>, time)
        AND views=<true, true, true>
        AND pct=percentageObject(Kaiku, 'any', O.cam, time)
ORDER BY pct DESC, time

```

Notice that *checkKindOfView* checks, with a single rendering, the kind of view that the TD has requested (i.e., viewing the front, top and side of the target). Besides, the function *percentageObject* obtains the percentage of the target that the camera will cover once it views the target (in *time* seconds).

4. Execution of the Query

Our proposal handles the processing of *location-dependent queries*, which are queries whose answer depends on the locations of the objects involved [3] (e.g., *obtain the cameras closer than 25 meters from the Kaiku rowing boat*). These queries are usually considered as *continuous queries* [30], whose answer must be continuously refreshed (in our tests, every second) due to the movements of the objects. In the scenarios we consider, not only the objects-of-interest can change their location but also the cameras (as they can be attached to moving objects). Moreover, the cameras can usually be rotated (pan and tilt); therefore, cameras that are not fulfilling currently the requirements of the TD could satisfy them in a near future, due to their combined change of location, pan, and tilt. In addition, the system is able to process multiple request at the same time; so, the TD can define a new query while the system is (continuously) executing others.

The most important task when executing a query is to analyze the views of the cameras (see Section 4.1). The goal of this analysis is to obtain high-level features of the camera view to check if it fulfills the TD requirements. The following high-level features are extracted by the system from a camera view:

- The specific objects viewed (e.g., in Fig. 5(a), *CAM1* views the rowing boats “Kaiku” –in green– and “Urdaibai” –in red–) and some information about them:
 - The distance to the object (e.g, in Fig. 5(a), the distance between *CAM1* and the boat “Urdaibai” is 17 meters).
 - The percentage of the object covered (e.g., in Fig. 5(b), *CAM2* views 22% of the boat “Kaiku”).
 - The percentage of the shot occupied by the object (e.g., in Fig. 5(b), “Kaiku” fills 26% of *CAM2* view).
 - The kind of view obtained of the object (e.g., in Fig. 5(b), *CAM2* views the front and left side of “Kaiku”).
 - The percentage of the viewpoint of the object covered (e.g., in Fig. 5(b), *CAM2* views 47% of the front and 29% of the left side of “Kaiku”).

- The percentage of the shot occupied by objects-of-interest (e.g., in Fig. 5(a), both rowing boats fill 6% of the view provided by *CAM1*).

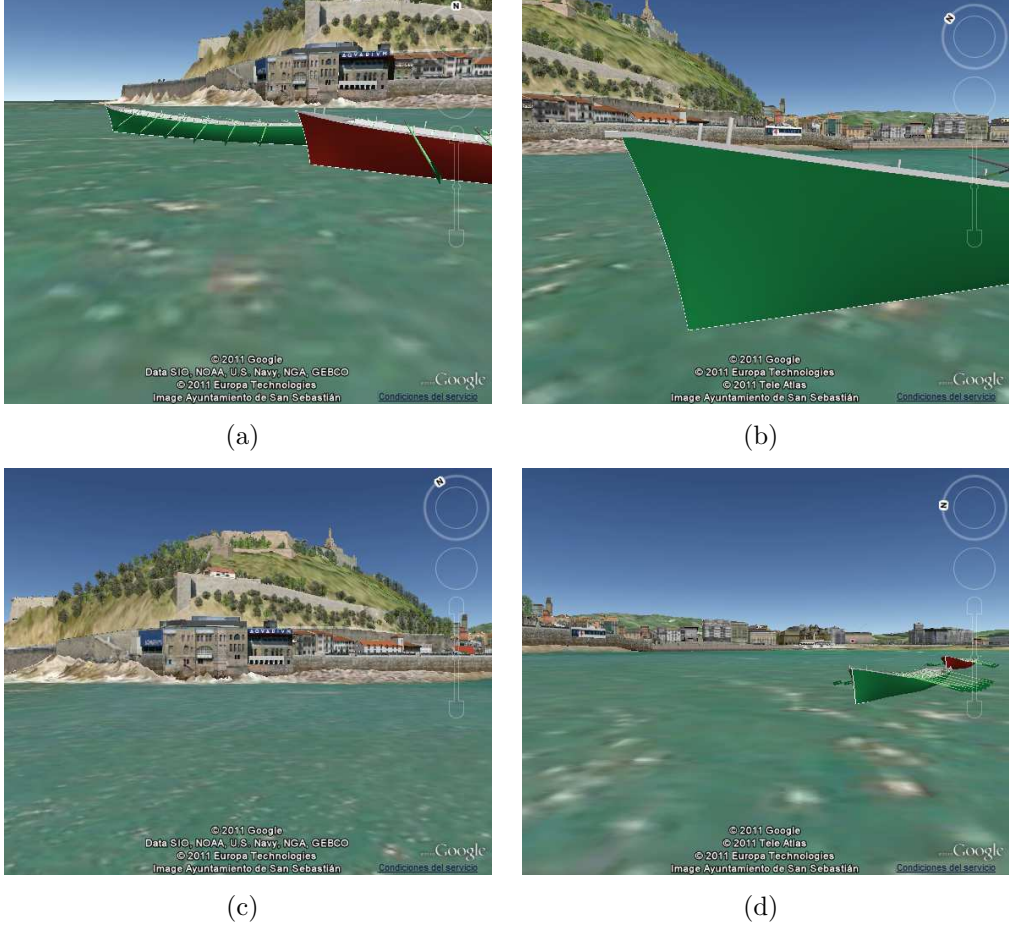


Figure 5: Sample views, recreated using Google Earth, of three cameras (*CAM1* (a), *CAM2* (b), *CAM3* (c), and *CAM3* after 4 seconds panning to the right (d)) covering a rowing race.

If a certain camera is not currently viewing a target object, the system can obtain the rotation (pan, tilt, or both) and the time needed for the camera to view it (see Section 4.2). For this purpose, the system needs to take into account the current location, speed, and direction of the target object and other objects in the scenario (because they could partially or fully hide the target), and the features of the camera being considered (maximum pan

and tilt allowed, and rotation speeds). Once the system estimates the time needed for a certain camera to view a target object, it will recreate the state of the scenario and obtain high-level features of the view that the camera would provide at that moment.

To illustrate the need of this time estimation and the rotations that should be performed to obtain a requested view, we present an example. Let us suppose that the TD is interested in: *cameras that can view the front, top, and side of a certain object, at most in 20 seconds, sorted by the percentage of the target viewed and the time needed to view it*. Using the location, speed, and direction of the objects and cameras in the scenario, the system estimates that *CAM3* (that is not currently viewing the target, see Fig. 5(c)) will need to pan 35 degrees (to the right) during 4 seconds to view the “Kaiku” boat. The view that it would obtain at that moment will cover the front, top, and side of “Kaiku” (see Fig. 5(d)), and so this camera fulfills the requirements of the TD. Notice, that *CAM1* also fulfills the requirements (see Fig. 5(a)) but *CAM2* does not view the top of “Kaiku” (see Fig. 5(b)).

The time and rotations estimated for a camera to fulfill the view required by the TD are high-level features obtained by the system. As in the case of the other high-level features, the time and rotations can be used as constraints in the user query and as ranking criteria when presenting the answer to the query.

4.1. Processing of the Camera Views

In this section, we explain how the system analyzes a camera view to obtain high-level features. First, we show how the viewpoint of the target that the camera is capturing is obtained. Then, we describe how the percentage of the target object viewed by a camera is computed taking occlusions into account. Finally, we explain a combination of the two previous processes that allows obtaining the percentage of a specific viewpoint of an object that a camera is providing. These are extended explanations of the 3D operations that we briefly introduced in [31].

4.1.1. Kind of View Obtained of the Target Object

Being able to classify the video streams of the cameras according to the kind of view obtained enables the system to answer specific requests of the TD (e.g., *cameras viewing the front, top and right side of a certain object*). As the extent of the objects in the scene could be complex and we need to perform the calculations automatically and quickly (the 3D model of the scenario is

updated in our tests every second), we propose the use of light sources and illumination to calculate the kind of view that a camera is providing of an object (see Algorithm 1).

Algorithm 1 Calculate the kind of view obtained of a target object

Input: *target, cam, <views>*

Output: *<visible views>*

```

1: scene=recreate cam's view in the 3D engine
2: remove all the illumination sources of scene
3: for each object in the scene do
4:   if object == target then
5:     paint object with reflective texture
6:   else
7:     paint object in black (background color)
8:   end if
9: end for
10: for each view in <views> do
11:   create light source in view's direction
12:   set light source's color to an unused one from <red, blue, green>
13: end for
14: projection=obtain 2D projection of the scene
15: for each pixel in projection do
16:   if pixel's red, blue, or green channels  $\neq 0$  then
17:     set true in <visible views>[i] if <views>[i] light source's color ==
        pixel's color
18:   end if
19: end for
20: return <visible views>

```

The first step is to recreate the view of the camera in the 3D engine (we used in our prototype *JMonkeyEngine*³) by setting the virtual camera with the same location, direction, and Field of View (FOV) than the real one. Then, different colors are assigned to the target and other objects in the scenario so when the scene is illuminated only the parts of the target object that are not occluded by other objects will be visible. *Directional light sources*

³<http://jmonkeyengine.org/>

(which have no position –only a direction–, are considered “infinitely” far away, and send out parallel beams of light) are used to illuminate the parts of the object that belong to each requested view using different colors for each light source. In this way, the system checks several kinds of views with a single pass and efficiently decreases the number of renderings needed (obtaining a rendering is one of the most time-consuming tasks). For example, to check if the camera of Fig. 6(a) is viewing the top and rear of the boat, the system “selects” these parts of the object by using a red and a blue light source, respectively (see Fig. 6(b)). Finally, the systems checks the color of each pixel of the 2D projection of the 3D scene and if its equal to one of the colors used for the light sources that means that the camera is viewing, at least, some part of the target object belonging to the kind of view considered.

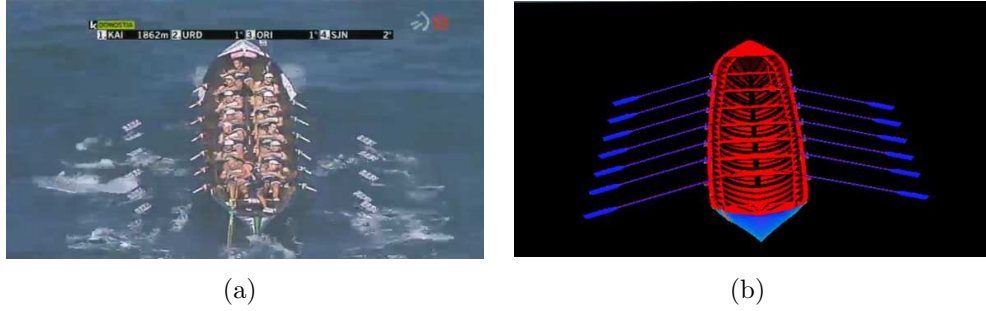


Figure 6: A real camera shot of a rowing boat (a) and the recreation of the view in our system with the top (red) and rear (blue) of the boat highlighted (b).

4.1.2. Percentage Viewed of the Target Object

Our system supports queries that ask for cameras that view a certain minimum percentage of an object. There exist two situations where a camera could have an incomplete view of an object: 1) when the target is partially or fully occluded by another object, and 2) when the target does not fit the FOV of the camera. Algorithm 2 calculates the percentage of a target object that a camera is viewing taking occlusions into account.

As in Algorithm 1, the system assigns different colors to the objects (red for the target object and transparent green for the other objects in the scenario), in order to show in the same rendering the hidden and visible parts of the target (see Fig. 7(b)). Then, the current FOV is painted in transparent blue to select what the camera is currently viewing (see Fig. 7(c)). If the target does not fit completely the FOV, the virtual camera is moved backwards

Algorithm 2 Calculate the percentage viewed of a target object

Input: *target, cam***Output:** *percentage viewed*

```
1: scene = recreate cam's view in the 3D engine
2: for each object in the scene do
3:   if object == target then
4:     paint object in red
5:   else
6:     paint object in transparent green
7:   end if
8: end for
9: paint current FOV in transparent blue
10: while target does not fit completely the FOV do
11:   move virtual camera backwards
12: end while
13: projection = obtain 2D projection of the scene
14: for each pixel in projection do
15:   if pixel's red channel  $\neq 0$  then
16:     increase #pixels of the target object
17:   end if
18:   if pixel's red and blue channels  $\neq 0$  and green channel == 0 then
19:     increase #pixels not occluded
20:   end if
21: end for
22: return  $\frac{\text{\#pixels not occluded}}{\text{\#pixels of the target object}}$ 
```

until it views the target object completely. This movement allows the system to obtain a rendering covering the full object while it does not affect the perspective of the scene (see Fig. 7(d)). Finally, the system obtains the total number of pixels of the target ($\#pixels\ of\ the\ target\ object$) and the pixels of the target visible and not occluded ($\#pixels\ not\ occluded$) and computes the percentage visible ($\frac{\#pixels\ not\ occluded}{\#pixels\ of\ the\ target\ object}$). For example, using the image of Fig. 7(d), the system obtains that the camera views 41% of the target object (the second boat).

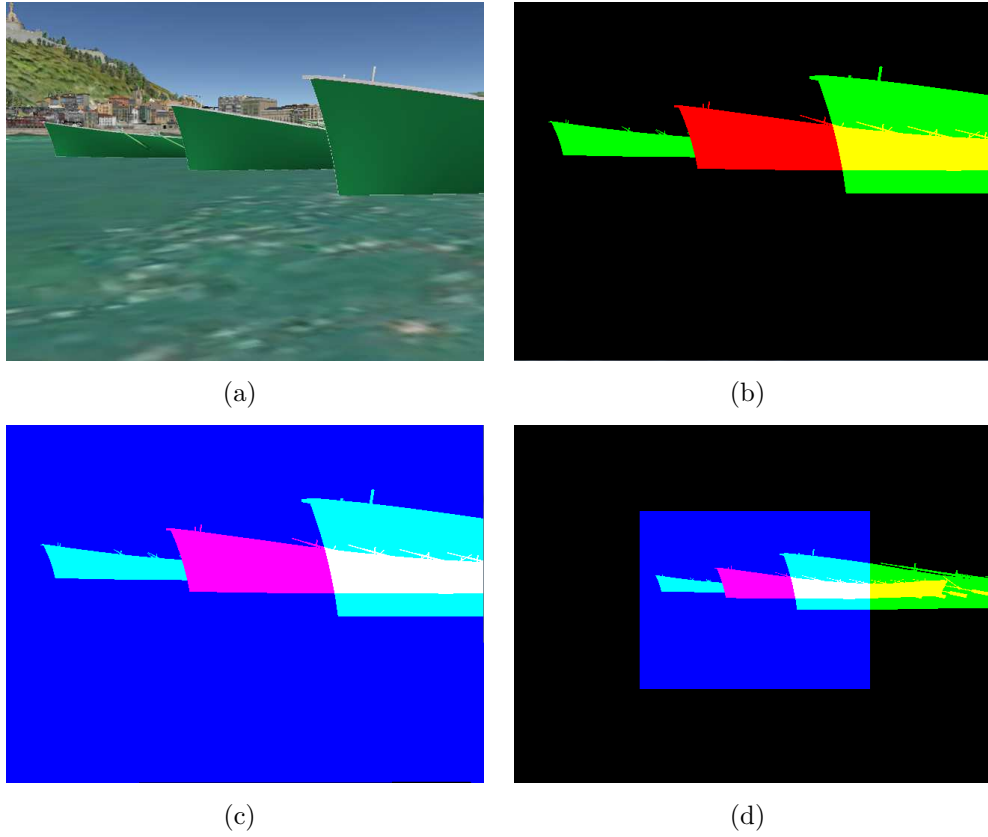


Figure 7: Computing the percentage of a target object in a shot: scene in *Google Earth* (a), selecting the target (b), painting the FOV (c), and covering the target completely (d).

4.1.3. Percentage of a Part of the Target Object

For the TD, it could be interesting also to retrieve cameras viewing a percentage of a certain part (i.e., top/bottom, front/rear, left/right) of an object (e.g., *cameras viewing at least 50% of the front of the object*). The method explained before needs an additional step to support this kind of queries (see Algorithm 3), to obtain first which shot would cover 100% of that target viewpoint in order to calculate the actual percentage viewed. In this way, the system sets the virtual camera of the 3D engine in the direction of the target viewpoint and at the same distance of the object than the real camera, and counts the number of illuminated pixels (*#pixels belonging to viewpoint*). This information will be used along with the number of pixels of the viewpoint that the camera is viewing (*#pixels viewed of viewpoint*) to calculate the percentage of the viewpoint viewed ($\frac{\text{\#pixels viewed of viewpoint}}{\text{\#pixels belonging to viewpoint}}$). Notice that, if the target object did not fit the FOV in Algorithm 3, the system moves the virtual camera backwards a distance d to compute the total amount of pixels visible for a shot that covers 100% of the target view. In this way, the system will move the camera backwards the same distance d before calculating the amount of pixels of the view that the camera covers. The example of Fig. 6(b) shows a 2D image rendered by the system for the current view of a camera, where the system obtains that the camera views 95% of the top and 92% of the rear of the target object. Notice that there are different intensities of red and blue in the image used by the system, as depending on the normal of the corresponding polygon the illumination method (*Phong* is used in JMonkeyEngine) makes it look darker or brighter. This is not a problem for our system because it only counts pixels that have a nonzero value for that specific channel.

4.2. Estimating Future Views Considering Object Trajectories

It could be interesting for a TD to show information about cameras that are not currently viewing a target object. For example, it could be useful to estimate if a camera is going to be able to view the target if rotated (and the rotation and time needed, if so). One can think that answering this question is easy; for example, if a camera has the object to its right then it should be able to view it if rotated to the right). However, calculating this estimation is not so easy when considering that objects in the scenario, and so also the cameras that are attached to them, can move. In the previous example, if the object keeps moving around the camera with a speed higher than the rotation

Algorithm 3 Calculate the number of visible pixels of the object in a shot that covers 100% of the target viewpoint of the object

Input: $target, view, cam$

Output: $pixels\ belonging\ to\ viewpoint, d$

```

1: recreate  $cam$ 's view in the 3D engine
2: paint  $target$  with reflective texture
3: set virtual camera's direction to  $view$ 's direction
4: create light source in  $view$ 's direction
5: if  $target$  does not fit completely the FOV then
6:    $d$ =move virtual camera backwards
7: end if
8:  $projection$ =obtain 2D projection of the scene
9:  $\#pixels\ belonging\ to\ viewpoint$ =count pixels in  $projection$ 
10: return  $\#pixels\ belonging\ to\ viewpoint, d$ 

```

speed of the camera, the camera will never view the target if rotated to the right. Thus, in the following we present in detail our approach to estimate the time and rotation needed by a camera to focus a target object considering that both objects and cameras can move.

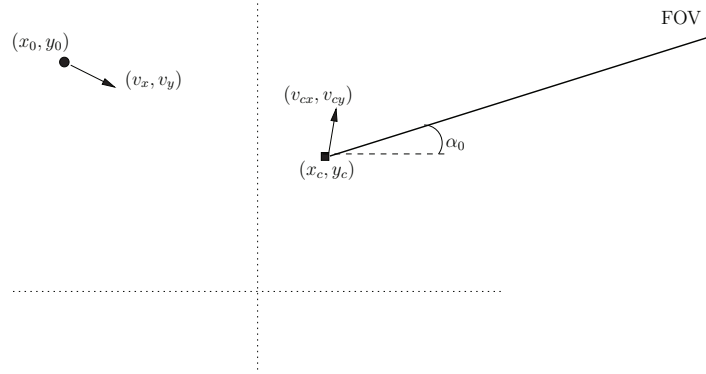


Figure 8: Initial state of a target object in (x_0, y_0) and a camera in (x_c, y_c)

On the one hand, we have to model the movement of the objects (that for simplicity can be represented here as points in the plane –their center

of mass—) with a motion function depending on time $S(t) = (x(t), y(t))$ (see Figure 4.2). For our scenario, we consider that the movement of the objects is linear, but the method presented in this section would work with any analytic functions $x(t)$, $y(t)$ (such as interpolation polynomials) or even with other approaches to model the movement of objects (e.g., [32, 33]). So, the motion function is:

$$S(t) = (x_0 + v_x t, y_0 + v_y t)$$

where (x_0, y_0) and (v_x, v_y) are the initial position and the speed vector of the object, respectively.

On the other hand, a camera is modeled as a semiline (defined by the bisector of its FOV) that can move and rotate (see Figure 4.2). The motion function for a camera is the semiline formed by the values of X and Y of the line $C(t) \equiv (X - x_c(t)) \sin(\alpha(t)) - (Y - y_c(t)) \cos(\alpha(t)) = 0$, such that $\text{sign}(X - x_c(t)) = \text{sign}(\sin(\alpha(t)))$, $\text{sign}(Y - y_c(t)) = \text{sign}(\cos(\alpha(t)))$, where $(x_c(t), y_c(t))$ is the translation motion function of the camera and $\alpha(t)$ is the rotation function.

Again, we consider that a camera has a linear translation motion with a uniform angular speed. Therefore, the motion equations of the semiline representing a camera depending on time are:

$$\begin{aligned} (X - (x_c + v_{xc}t)) \sin(\omega_c t + \alpha_0) - (Y - (y_c + v_{yc}t)) \cos(\omega_c t + \alpha_0) &= 0 \\ \text{sign}(X - (x_c + v_{xc}t)) &= \text{sign}(\sin(\omega_c t + \alpha_0)) \\ \text{sign}(Y - (y_c + v_{yc}t)) &= \text{sign}(\cos(\omega_c t + \alpha_0)), \end{aligned}$$

where (x_c, y_c) and (v_{xc}, v_{yc}) are the initial position and the speed vector of the camera, respectively, ω_c is the pan speed of the camera, and α_0 is the initial pan of the camera.

We want to obtain the minimum time instant when the camera focuses the object, considering that the camera can rotate to the left side (a positive pan speed) or to the right side (a negative pan speed). Thus, we have to obtain:

- which pan speed (positive or negative) leads to a faster movement to focus the object,

- for that pan speed, the time instant when the camera and the object intersect.

In order to compute these values, first we have to find the solution t_{f+} for the equation

$$E(t) = (x_0 + v_x t - (x_c + v_{xc} t)) \sin(\omega_c t + \alpha_0) - (y_0 + v_y t - (y_c + v_{yc} t)) \cos(\omega_c t + \alpha_0) = 0$$

that holds

$$t_{f+} = \min\{t_r | t_r \geq 0, E(t_r) = 0, \\ \text{sign}(x_0 + v_x t_r - (x_c + v_{xc} t_r)) = \text{sign}(\sin(\omega_c t_r + \alpha_0)), \\ \text{sign}(y_0 + v_y t_r - (y_c + v_{yc} t_r)) = \text{sign}(\cos(\omega_c t_r + \alpha_0))\}$$

Second, we have to find the solution t_{f-} changing ω_c by $-\omega_c$ in the above equation. The minimum value of $\{t_{f+}, t_{f-}\}$ gives us the sign of the pan speed and the time instant that we are looking for (see Figure 9).



Figure 9: t_{f+} (t_{f-}) time to focus the target object rotating the camera to the left (right) side.

The trajectories of the objects are estimated by using linear extrapolation based on their speed vectors. The speed vector of an object is computed by

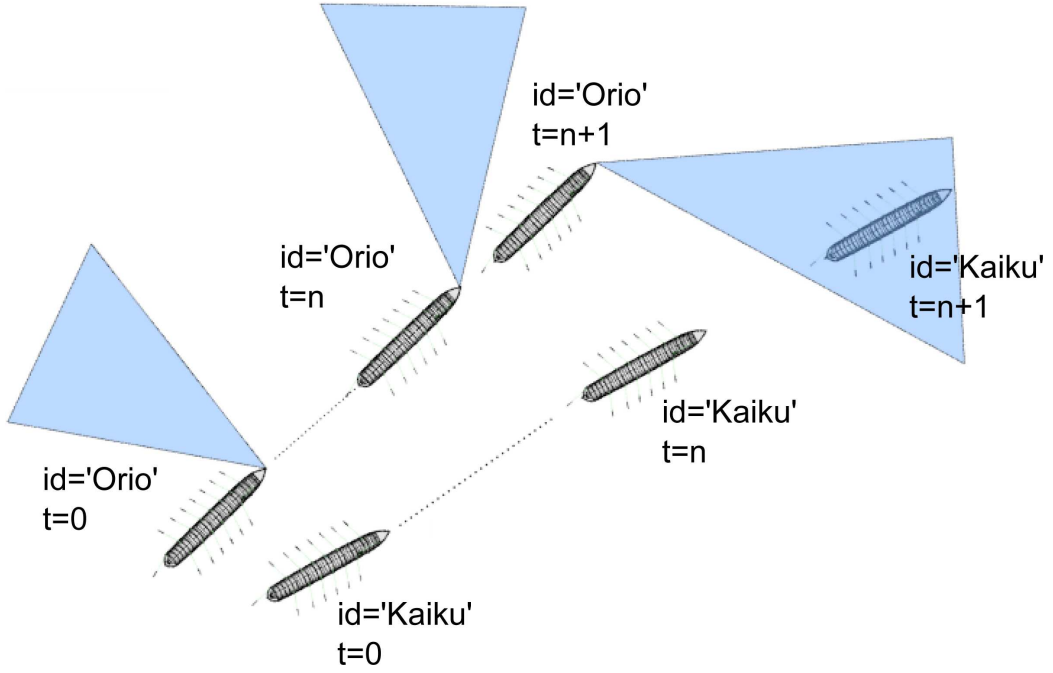


Figure 10: Estimation of trajectories and time needed to focus “Kaiku” from “Orio”

considering three previous reference locations of the object (in our prototype, the locations of the objects during the last three seconds). To solve the above equation we reduce the problem to finding the zeros of a polynomial. We use an approximation of \sin and \cos using Taylor polynomials and solve numerically the equation using Laguerre’s method, a root-finding algorithm tailored to polynomials. As an example, Figure 10 shows the estimation of the time needed to focus horizontally a target object by considering the movements of the objects, the current pan, and the pan speed of the camera.

As we are dealing with a scenario where the objects can move in 3D, we need to obtain the time needed to focus the target both horizontally (pan) and vertically (tilt). As pan and tilt movements can be done in parallel, we obtain the maximum of the time needed to pan and the time needed to tilt and use that value as the time needed to focus the object. So, the equation above is used for both movements using the horizontal plane for the pan (equation above) and the plane defined by the trajectory of the object and the z-axis for the tilt.

Once the estimation has been completed, the system generates the scene

to calculate if the requirements of the query (e.g., type of view, percentage of the target object or part shown, etc.) would be satisfied. Notice that, as the objects are moving, when the camera is able to focus the target it could happen to be occluded by another object. Therefore, the trajectories of all the objects have to be taken into account too, not only the trajectory of the camera and the target object.

5. GUI: obtaining the user requirements and presenting the results

Inspired by mobile production units, we have developed a friendly GUI that models the TD work environment (see Fig. 11 and <http://sid.cps.unizar.es/MultiCAMBA/>) where the user can express his/her requirements and the results are displayed [34]. The GUI is mainly composed of three modules:

1. The *query interface*, where the TD defines (using HTML forms) the requirements that the cameras have to fulfill and stores/loads/submits his/her queries.
2. The *overview map*, which is a 3D representation of the scenario, with the moving objects and cameras involved, where the results of the queries are shown.
3. The *camera inputs*, which are several windows where the TD can preview the camera video streams before broadcasting them.

The results obtained by the system (the cameras fulfilling the TD requirements) are displayed in a tabular form in the query interface. The results can be sorted according to any of the high-level features extracted (see Section 4).

Delivering the information easily and effectively is essential to quickly select the camera to broadcast in live. To achieve this goal we use a powerful and free software tool, the *Google Earth API*, to display the results in a friendly interface. Google Earth is a geographic information system that offers a vast amount of geospatial data (satellite images, 3D buildings, 3D terrains, etc.), that helps to develop virtual scenes similar to those in the real world. Thanks to this, the overview map recreates and keeps up-to-date the scene in a Google Earth plugin allowing the TD to navigate through the scenario. In the center of Fig. 11 the overview map shows an example of the moving objects and cameras (a brown triangle indicates its current FOV) in a sport scenario. Besides, it shows the results to a query submitted by the TD



Figure 11: Graphical User Interface (GUI) for the Technical Director.

to retrieve the cameras that can view a certain object (a yellow star is used to represent the target object, a green hexagon for the cameras fulfilling the requirements, a blue hexagon for the cameras that will fulfill them if rotated, and a red hexagon for the cameras unable to fulfill them).

The system also uses Google Earth to recreate the view of a camera. This is very useful mainly in two situations: when the real camera video stream is not available (as in the camera inputs of Fig. 11) and when a camera is not currently viewing the target and the system estimates the scene it will capture if it is rotated. The possibility to estimate future camera views with the combined use of Google Earth technology allows the system to show a realistic recreation of what a camera will view if rotated. This is interesting because sometimes the best shot is not the one that can be obtained the fastest. For example, a camera that is not currently viewing a target, but will be in a matter of seconds, could then provide a better background scene than a camera that is viewing the target currently, or could cover a greater percentage of the target, as in the last sample query of Section 3.2 (notice that in that sample query the system will show to the TD the image of Fig. 5(d) as a preview of the future view).

Once the cameras that obtain the view interesting for the TD are displayed in the GUI, he/she has enough information to select the camera whose view will be broadcasted. He/she could also request the camera operator to rotate cameras as suggested by the system. Notice that, with the information provided by the system about the required turning (pan, tilt, or both) needed by a camera to view a certain target object, it could be possible for the system to automatically control the cameras to track a target and keep it centered in the FOV, if ordered by the TD.

6. Experimental Evaluation

In this section, we show the experimental evaluation performed to validate our system. An experimental comparison of our system with other similar approaches [4, 5] is not feasible because of the great difference on both the use cases considered (soccer in [4, 5] vs. rowing boat races in our system) and the input data that the systems need (their systems use real videos while ours uses information about the location and direction of objects and cameras). Therefore, we decided to focus the experiments on the evaluation of our system in the rowing race scenario, considered as a use case and explained along the paper. More in detail, in this section we first explain the prototype of the system developed and how the simulation of a real rowing race has been carried out. Then, we show the experiments performed to test: 1) the quality of the result set, 2) the precision of the estimated time to view an object, 3) the precision of the estimated percentage viewed of an object, and 4) the behavior of the system compared with real camera footage⁴.

6.1. Prototype of the System

We developed a prototype of the system to perform an experimental evaluation of our approach (see Figure 12). Regarding the details of the implementation, this prototype has been developed as a Web application using HTML5 because the latest Google Earth API (we explained the importance of this technology for our system in Section 5) is based on JavaScript and meant to be used in web pages. Then, the core of our system has been developed as a Java Applet that has been integrated into the Web application. We selected the Java programming language because for the processing of

⁴Some videos and interesting moments of the tests are available at <http://sid.cps.unizar.es/MultiCAMBA/Experiments>.

the camera views we are using a free and powerful Java 3D engine: JMonkeyEngine (as we explained in Section 4.1). Both JMonkeyEngine and the Google Earth plugins extract the 3D meshes of objects-of-interest from OBJ (a geometry definition format) files. Also, we are using a MySQL database to store the 3D model of the scenario and other interesting information for the tests.

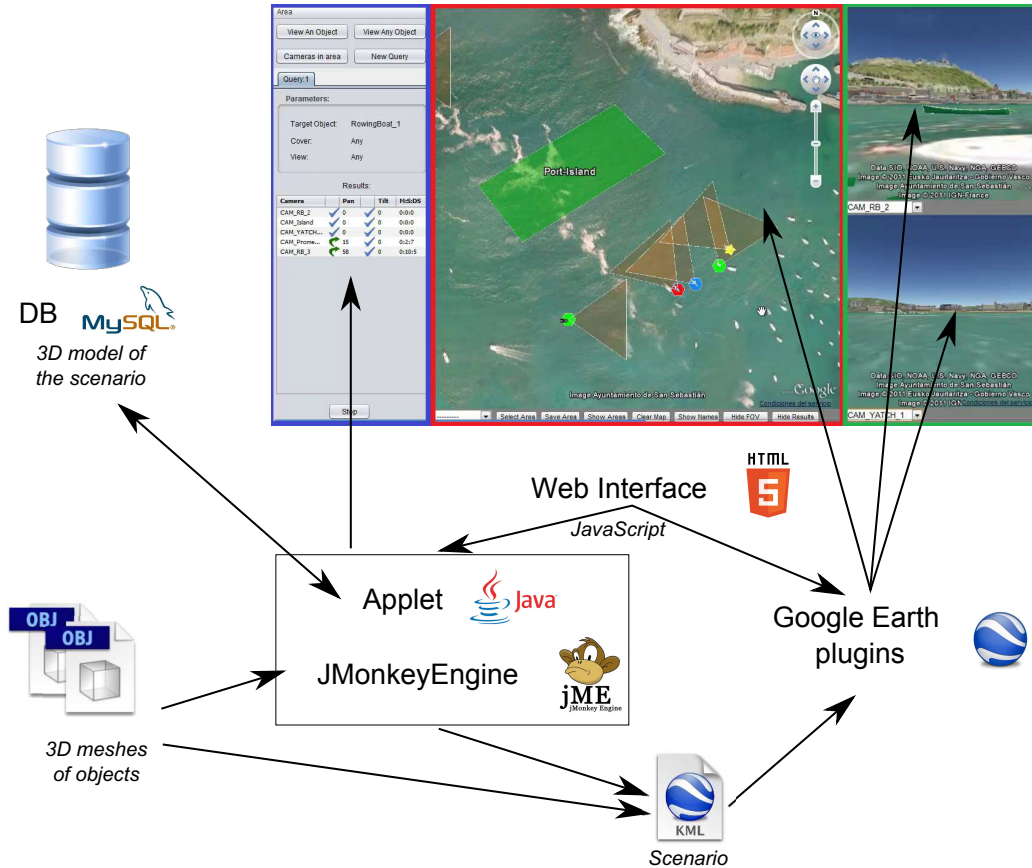


Figure 12: Technical architecture diagram of the prototype of the system.

We need to enable the communication between the different technologies and the transfer of information among them. For example, for the communication among the applet and the Google Earth plugins our prototype uses the Keyhole Markup Language (KML)⁵. For this task, the applet creates and

⁵<https://developers.google.com/kml>

maintains updated a KML file with the current location, direction, and other information of cameras and objects in the scenario, that all the Google Earth plugins use to update the scene they show.

6.2. *Simulating the Scenario*

Testing a live broadcasting of a rowing race in a real-life environment is difficult because there are many real objects, devices, and wide-area scenarios involved. Besides, testing the system several times in similar situations in a real-life environment is challenging. Therefore, we have developed a simulator that enables us to manage the different cameras and objects in the scenario. To simulate the rowing race we have used a file containing the *real* GPS location data of each rowing boat captured every second during the race celebrated in San Sebastian in September 2010; this rowing race covers a total distance of 3 miles logically divided in two parts by a turning point. Therefore, *real* trajectories are used to move objects in the simulations. Moreover, the simulator allows us to dynamically change other parameters, such as the current pan and tilt of each camera, in order to rotate them as the TD would request in the real scenario.

The parameters used in the tests are the following ones:

1. There are four rowing boats equipped with a camera; as commented before, these boats move according to the real GPS location data captured during the race celebrated in San Sebastian in September 2010.
2. There are three other cameras: one on top of the island, one on the promenade, and one on a sailing boat near the rowing boats. The cameras are set with horizontal focus $\beta_h = 70^\circ$, vertical focus $\beta_v = 45^\circ$, pan range $\pm 130^\circ$, tilt range $\pm 90^\circ$, and pan and tilt speed 5.5 degrees/second.
3. The tests were performed in an Intel Core i5-480M with graphics card NVIDIA Geforce GT 540M.

For the experimental evaluation we will consider this simulated scenario and one of the most interesting queries for the TD. In our sample scenario, the TD may want to know, during the whole event, which cameras are viewing each of the rowing boats, as he/she could need a shot of a certain boat at anytime. In fact, during the broadcasting of any event, the TD would be interested on the cameras that are viewing the main agents (e.g., for a soccer match it is interesting to know the cameras that are viewing each of the star

players). So, as a representative query in the context of our experiment we will continuously process the following one:

“Cameras that can view at least 70% of the Kaiku boat”

6.3. Evaluating the Quality of the Result Set

In this first experiment, we want to evaluate whether the cameras provided by the system as an answer are good candidates to provide the views required by the TD (see Fig. 13). We represent the number of cameras in the answer (vertical axis) along the event duration (horizontal axis). The blue line shows the number of cameras provided by the system (some of them currently fulfilling the requirements of the user and the others estimated by the system to be able to do it in the near future) and the red dashed line shows the number of wrongly chosen cameras. We consider a camera as wrongly chosen when the system makes an estimation error greater than 25 seconds in the estimated time to wait until the camera could provide the view required. For example, at the beginning the system estimates that the camera on board the farthest boat (that has two boats between it and the target and does not currently view the target) will cover 77% of “Kaiku” in 10 seconds, but when those 10 seconds have passed the camera is only able to view 21% because then the occlusion of “Kaiku” is greater than estimated. This occlusion remains for 30 seconds, and so the system makes a mistake considering this camera as part of the answer. These kinds of errors only happen when the target is occluded due to variations in the speed and direction of the objects that make the system fail in the estimation of the future scene. Again, around time 17:30, where the boats are in the final sprint, the system estimates the time needed to view the target and shows the cameras in the answer, but some of them overtake “Kaiku” and thus they are not able to view it for the rest of the race (due to their rotation limits). However, thanks to the continuous query processing, estimation errors due to unexpected changes in the trajectories are quickly corrected.

For the test, the cameras are rotated automatically in order to track “Kaiku” according to the results provided by the system. The maximum number of cameras that can be part of the answer is six (because the camera of “Kaiku” cannot view itself due to the physical rotation limits), and the system shows in the answer at least three cameras during the most critical time intervals, that is, when the boats are at the turning point (which is a key moment during the race) around time instant 10:30 and when the other boats are overtaking “Kaiku” around time instant 17:00-20:00.

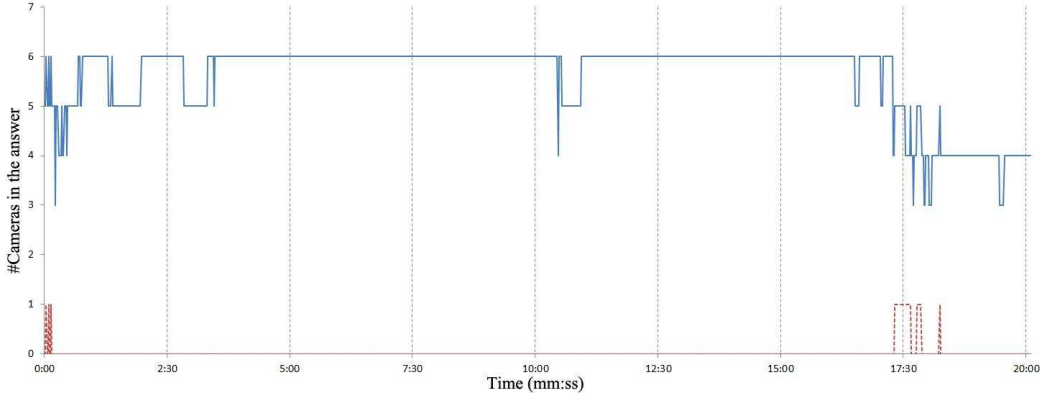


Figure 13: Quality of the result set (i.e., cameras fulfilling the user requirements): number of cameras in the answer set (blue line) and number of wrongly chosen cameras (red dashed line).

Even though the system must perform its calculations every second, the results are quite satisfactory for the total period of twenty minutes, and the errors are corrected quickly enough to avoid a negative impact on the decisions of the TD. Notice that, in our scenario, for the most part of the race a good number of cameras fulfill the TD requirements, as they are rotated automatically following the system commands. In this case, the system succeeds in discarding the irrelevant cameras at each moment and in ranking the most interesting cameras first.

6.4. Testing the Precision of the Estimated Time to View

The goal of this test is to evaluate the error in the time estimation, measured in seconds between the estimated time and the actual time when the camera views the target.

In Fig. 14, we show the sum of the time errors for all the cameras at every time instant. Notice that there are positive and negative values, that represent when the time estimated by the system is greater than the actual value (positive) or when it is smaller (negative). We have decided to make this distinction because a negative error could make the TD to keep an eye on a camera that actually will need more time than expected to view the target, which may be an important problem. On the other hand, a positive error (if it is not too big) means that a camera viewed the target earlier than expected, and so the negative impact of selecting that camera is minimized. Anyway,

notice that the sum of all the errors ranges only between -5 and $+3$ seconds, which is very small for the sum of the errors of all the cameras.

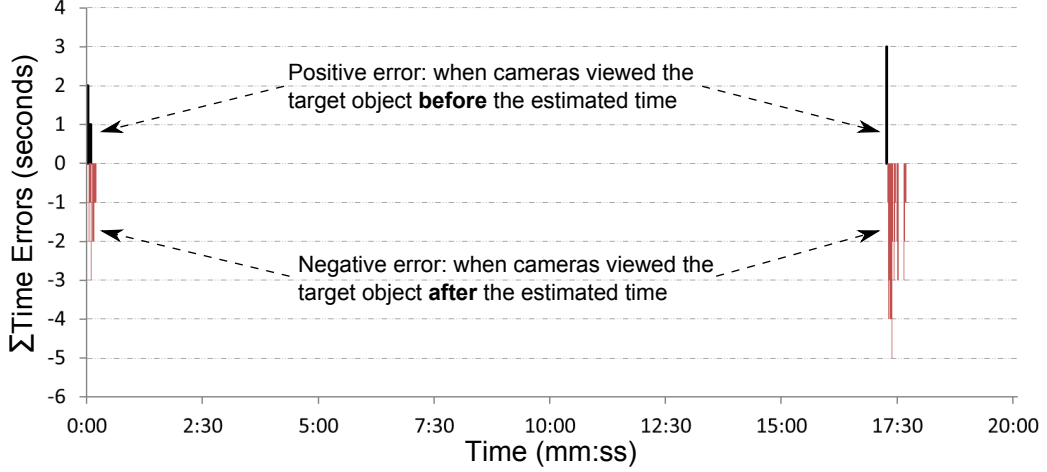


Figure 14: Error in the estimated time needed for the cameras to view the target object: the error is localized at two specific time intervals (the start and the end of the race).

The errors are localized within two specific time intervals. The first errors occur at the start of the race, when all the cameras are pointing at the same direction as the front vector of their rowing boats. In this scenario there are not big changes on the objects' altitude (only slight changes caused by waves), so in order to test the pan and tilt estimation we have set initially all the cameras pointing upwards at the start (maximum tilt). Thus, as the test starts, the cameras have to pan and tilt to view "Kaiku" and the system has to estimate the time needed to do it. As at the starting point the system has not enough information to precisely estimate the speeds of the boats, it makes some little mistakes that, overall, do not exceed 3 seconds. Thus, this error is small enough to provide the TD with accurate information. Around time 17:30 the end of the race is near and there are big variations in the speed and distance between the boats (as the rowers are making their last efforts) and some boat is even overtaken. The system estimates here that the time needed to view "Kaiku" is smaller than the actual time needed, as the boats increase their speed in a final attempt to win the race.

This test shows that the errors concerning the estimated time to view the target are small. Besides, those errors are localized in two narrow time intervals and they are quickly fixed (in the test, in ten seconds at most),

since the corresponding location-dependent query is continuously processed.

6.5. Testing the Precision of the Estimated Percentage Viewed

We also tested the precision of the estimated percentage of the target viewed by the cameras. Due to the specific features of the scenario, a camera views a percentage below 100% when the target is partially occluded by another boat (the target usually remains inside the FOV of the cameras). As explained before, the system is able to compute the percentage of “Kaiku” that a camera will cover when it is able to view it. For the camera closest to “Kaiku” (that has no other boat between them), the errors in the estimation of the percentage only occur when there is an error in the estimation of the time needed to view it. We have considered these errors in Fig. 14. We show in Fig. 15 an example for a camera that has two boats between it and the target (and thus occlusions are possible) and focusing on a short time interval (the first 8 seconds) where some errors occur. The system starts estimating that the camera will view a smaller percentage than what it will really view, and as the time goes by the error in this estimation decreases.

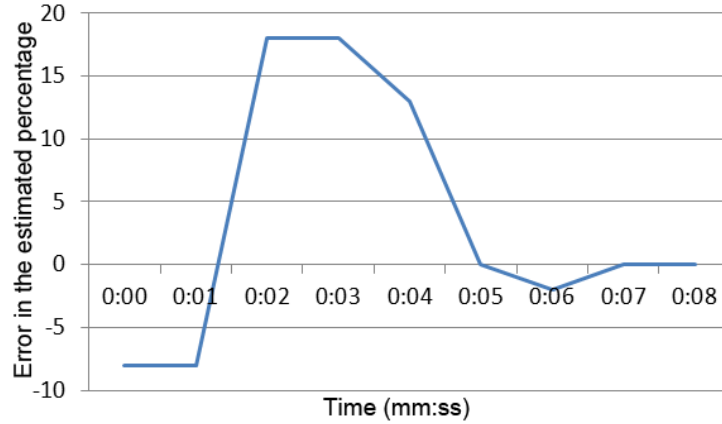


Figure 15: Error in the estimated percentage of the target that a camera will view in the first eight seconds.

6.6. Testing the System Against Real Camera Footage

We have also tested the system against real camera footage. Specifically, we have used the video produced by the Spanish broadcaster *EiTB* for the rowing race celebrated in September 2010. The goal of this test was to

compare the results offered by our system with the real event to check the behavior of our proposal. Fig. 16 shows an extract of two seconds of a camera covering the end of the race (the real images are shown on the bottom of each frame) compared with the information generated and exploited by the query processing in our system (shown at the top of each frame). Notice that the extents of the rowing boats are represented in green by the system, for an easy comparison with the real footage. At the beginning of this live footage, the “Urdaibai” boat has just crossed the finish line (it is the only boat we see in the first frame in Fig. 16(a)); it is followed by “Kaiku”, that enters the FOV of the camera 1 second later, being completely captured by the camera exactly at time instant 2.08 seconds (Fig. 16(b)).



Figure 16: Testing the system against real camera footage (the information generated by our system is on the top): in two consecutive seconds.

At the time instant when the live footage begins, our system estimates that the camera will obtain a full view of the “Kaiku” boat in 1.85 seconds. Notice that the error committed is not very significant: the TD will be alerted just 0.23 seconds before the desired situation is captured by the real camera. This small error is caused by the slight inaccuracy of the GPS data transmitted by the boats and the fact that the location of the real camera was estimated (the TV broadcaster did not provide us with this information). However, the results obtained by the system would have been good enough to help the TD to select this camera to view the “Kaiku” boat.

7. Conclusions

In a live broadcasting, the higher the number of cameras available, the more difficult the selection of the camera whose view must be broadcasted.

So, in order to alleviate this task, we have developed a system that allows a Technical Director (TD) to express his/her interests in specific camera views, and obtains automatically a sorted set of cameras that could provide such views. The main features of our proposal are:

- The system analyzes camera views in real-time by considering a 3D model of the scenario updated continuously with the information of objects and cameras involved. Besides, it provides a good performance in scenarios where the target object is partially or totally hidden by other objects, as it takes occlusions into account.
- The system obtains high-level features of a camera view, such as: the specific objects viewed, the percentage of them covered, the percentage of the shot filled by a specific object, the kind of view of the object obtained (e.g., front, top, side), etc.
- The system generates location-dependent queries, using the information provided by the TD, and continuously processes them to obtain an accurate answer. In this way, our proposal supports highly-dynamic scenarios.
- The system performs efficiently. The experimental results show the feasibility of our proposal, that can be used for the real-time selection of cameras. It executes a query in less than a half second.
- The system provides a user-friendly GUI that allows the TD to easily indicate the specific view he/she is interested in and presents the results to the queries.

So, as long as the location of the objects-of-interest and cameras (and an approximation of the extent of the objects) can be obtained in real-time, the system presented can be applied to any context. In some situations it could be challenging to obtain this information for certain objects; however, our approach does not rely on a specific technology nor requires 100% precision of these data to effectively distinguish between cameras that are interesting or not for a given query and to facilitate the selection of the TD by providing a ranking of the candidate cameras.

As future work, we plan to provide the system with the capability to manage knowledge related to cinematic shots and scenarios. In this way, TDs could use technical language to communicate the view they are interested in (e.g., a “high-angle shot” of the first boat to cross the finish line).

Acknowledgments.

This work has been supported by the CICYT project TIN2010-21387-C02 and DGA-FSE. We thank David Antón and Francisco J. Serón for their help with the implementation of our prototype and technical support, respectively.

References

- [1] Y. Chen, H. Sampathkumar, B. Luo, X. Chen, iLike: Bridging the semantic gap in vertical image search by integrating text and visual features, *IEEE Transactions on Knowledge and Data Engineering* PP (99) (2012) 1.
- [2] Y. Yildirim, A. Yazici, T. Yilmaz, Automatic semantic content extraction in videos using a fuzzy ontology and rule-based model, *IEEE Transactions on Knowledge and Data Engineering* 25 (1) (2013) 47–61. doi:<http://doi.ieeecomputersociety.org/10.1109/TKDE.2011.189>.
- [3] S. Ilarri, E. Mena, A. Illarramendi, Location-dependent query processing: Where we are and where we are heading, *ACM Computing Surveys* 42 (3) (2010) 1–73.
- [4] J. Wang, C. Xu, E. Chng, H. Lu, Q. Tian, Automatic composition of broadcast sports video, *Multimedia Systems* 14 (4) (2008) 179–193.
- [5] K. Choi, S. Lee, S. Y., Automatic broadcast video generation for ball sports from multiple view, in: *International Workshop on Advanced Image Technology (IWAIT'09)*, 2009.
- [6] F. Chen, C. D. Vleeschouwer, Personalized production of basketball videos from multi-sensored data under limited display resolution, *Computer Vision and Image Understanding* 114 (6) (2010) 667–680.
- [7] A. Ekin, A. M. Tekalp, R. Mehrotra, Automatic soccer video analysis and summarization, *IEEE Transactions on Image Processing* 12 (7) (2003) 796–807.
- [8] N. Nitta, Y. Takahashi, N. Babaguchi, Automatic personalized video abstraction for sports videos using metadata, *Multimedia Tools and Applications* 41 (1) (2009) 1–25.
- [9] M. H. Kolekar, Bayesian belief network based broadcast sports video indexing, *Multimedia Tools and Applications* 54 (2011) 27–54.
- [10] C.-C. Cheng, C.-T. Hsu, Fusion of audio and motion information on HMM-based highlight extraction for baseball games, *IEEE Transactions on Multimedia* 8 (3) (2006) 585–599.
- [11] N. Babaguchi, Y. Kawai, T. Ogura, T. Kitahashi, Personalized abstraction of broadcasted American football video by highlight selection, *IEEE Transactions on Multimedia* 6 (4) (2004) 575–586.

- [12] C. Xu, Y.-F. Zhang, G. Zhu, Y. Rui, H. Lu, Q. Huang, Using webcast text for semantic event detection in broadcast sports video, *IEEE Transactions on Multimedia* 10 (7) (2008) 1342–1355.
- [13] C. Xu, J. Wang, H. Lu, Y. Zhang, A novel framework for semantic annotation and personalized retrieval of sports video, *IEEE Transactions on Multimedia* 10 (3) (2008) 421–436.
- [14] G. Zhu, Q. Huang, C. Xu, L. Xing, W. Gao, H. Yao, Human behavior analysis for highlight ranking in broadcast racket sports video, *IEEE Transactions on Multimedia* 9 (6) (2007) 1167–1182.
- [15] F. Chen, D. Delannay, C. D. Vleeschouwer, An autonomous framework to produce and distribute personalized team-sport video summaries: a basket-ball case study, *IEEE Transactions on Multimedia* 13 (6) (2011) 1381–1394.
- [16] T. D’Orazio, M. Leo, A review of vision-based systems for soccer video analysis, *Pattern Recognition* 43 (2010) 2911–2926.
- [17] M. H. Bianchi, Automatic video production of lectures using an intelligent and aware environment, in: *Third International Conference on Mobile and Ubiquitous Multimedia (MUM’04)*, ACM, 2004, pp. 117–123.
- [18] Y. Rui, L. He, A. Gupta, Q. Liu, Building an intelligent camera management system, in: *Ninth ACM International Conference on Multimedia (MULTIMEDIA’01)*, ACM, 2001, pp. 2–11.
- [19] Q. Liu, D. Kimber, J. Foote, L. Wilcox, J. Boreczky, FlySPEC: a multi-user video camera system with hybrid human and automatic control, in: *Tenth ACM International Conference on Multimedia (MULTIMEDIA’02)*, ACM, 2002, pp. 484–492.
- [20] C. Zhang, Y. Rui, J. Crawford, L.-W. He, An automated end-to-end lecture capture and broadcasting system, *ACM Transactions on Multimedia Computing, Communications and Applications* 4 (1) (2008) 6:1–6:23.
- [21] R. Heck, M. Wallick, M. Gleicher, Virtual videography, *ACM Transactions on Multimedia Computing, Communications, and Applications* 3 (1), 28 pages.
- [22] D. G. Aliaga, Y. Xu, V. Popescu, Lag camera: A moving multi-camera array for scene-acquisition, *Journal of Virtual Reality and Broadcasting* 3 (10).
- [23] H.-S. Park, S. Lim, J.-K. Min, S.-B. Cho, Optimal view selection and event retrieval in multi-camera office environment, *Multisensor Fusion and Integration for Intelligent Systems* 35 (2009) 45–53.
- [24] R. Yus, S. Ilarri, E. Mena, Real-time selection of video streams for live TV broadcasting based on query-by-example using a 3D model, *Multimedia Tools and Applications* Published online: June 2013, 27 pages, DOI: <http://dx.doi.org/10.1007/s11042-013-1550-5>.

- [25] J. Cha, M. Eid, A. E. Saddik, Touchable 3D video system, *ACM Transactions on Multimedia Computing, Communications, and Applications* 5 (4) (2009) 29:1–29:25.
- [26] K. Serr, T. Windholz, K. Weber, Comparing GPS receivers: a field study, *URISA Journal* 18 (2).
- [27] T. Ansary, M. Daoudi, J.-P. Vandeborre, A bayesian 3-D search engine using adaptive views clustering, *IEEE Transactions on Multimedia* 9 (1) (2007) 78–88.
- [28] Y. Gao, M. Wang, Z.-J. Zha, Q. Tian, Q. Dai, N. Zhang, Less is more: Efficient 3-D object retrieval with query view selection, *IEEE Transactions on Multimedia* 13 (5) (2011) 1007–1018.
- [29] S. Ilarri, E. Mena, A. Illarramendi, R. Yus, M. Laka, G. Marcos, A friendly location-aware system to facilitate the work of technical directors when broadcasting sport events, *Mobile Information Systems* 8 (1) (2012) 17–43.
- [30] D. Terry, D. Goldberg, D. Nichols, B. Oki, Continuous queries over append-only databases, *ACM SIGMOD Record* 21 (2) (1992) 321–330.
- [31] R. Yus, E. Mena, J. Bernad, S. Ilarri, A. Illarramendi, Location-aware system based on a dynamic 3D model to help in live broadcasting of sport events, in: *19th ACM International Conference on Multimedia (MM 2011)*, ACM, 2011, pp. 1005–1008.
- [32] Y. Tao, C. Faloutsos, D. Papadias, B. Liu, Prediction and indexing of moving objects with unknown motion patterns, in: *Proceedings of the 2004 ACM SIGMOD international conference on Management of data, SIGMOD '04*, 2004, pp. 611–622.
- [33] A. M. Hendawi, M. F. Mokbel, Predictive spatio-temporal queries: a comprehensive survey and future directions, in: *Proceedings of the First ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems, MobiGIS '12*, 2012, pp. 97–104.
- [34] R. Yus, D. Anton, E. Mena, S. Ilarri, A. Illarramendi, MultiCAMBA: A system to assist in the broadcasting of sport events, in: *Eighth Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQ-uitous 2011)*, Springer, 2011, pp. 238–242.