LiveSync: a Tool for Real Time Video Streaming Synchronization from Independent Sources

Marcello N. de Amorim[1], Ricardo M. C. Segundo[2], Celso A. S. Santos[3]

Universidade Federal do Espirito Santo, Brazil
novaes@inf.ufes.br , rmcs87@gmail.com , seibel@inf.ufes.br

Abstract. This work presents a tool that allows users to synchronize live videos from multiple sources such as YouTube or any other video streaming sources. The proposed approach to proceed the multiple camera video synchronization is based in crowdsourcing techniques, using the power of a crowd of collaborators to synchronize videos, requiring from each user the sync of only a pairs of videos. Additional sync relations are inferred from the known contributions, using transitivity properties and an appropriate structure for this inference, the Dynamic Alignment List.

User Generated Videos are contents created by heterogeneous users around an event. Each user films the event with his point of view, and according to his limitations. In this scenario, it is impossible to guarantee that all the videos will be stable, focused on a point of the event or other characteristics that turn the automatic video synchronization process possible. Focused on this scenario we propose the use of crowdsourcing techniques in video synchronization (CrowdSync). The crowd is not affected by heterogeneous videos as the automatic processes are, so it is possible to use them to process videos and find the synchronization points. In order to make this process possible, a structure is described that can manage both crowd and video synchronization: the Dynamic Alignment List (DAL). Therefore, we carried out three experiments to verify that the crowd can perform the proposed approach: the first experiment used a crowd simulator to verify the DAL capability of managing videos and contributions, generating cohesive video presentations; the second experiment used a crowd to synchronize videos performing small tasks; the third explored the use of the crowd to synchronize Live Stream Videos, through the development of the LiveSync tool.

Categories and Subject Descriptors: H.4.4.3 [**Information Systems Applications**]: Crowdsourcing; H.3.2.7 [**Human-Centered Computing**]: Synchronous editors

Keywords: live video, synchronization, crowdsourcing

1. INTRODUCTION

Multiple camera video synchronization is a research area within multimedia. Automatic video synchronization (AVS) is a form to synchronize multiple video streams. AVS can be done analysing video segments [Wang et al. 2014] or audio ones [Su et al. 2012]. In Schweiger et.al.[Schweiger et al. 2013] we find these and other approaches in the area. One main contribution of this paper is the description of the challenges for automatic synchronization algorithms: wide baselines, camera motion, dynamic backgrounds and Occlusions.

We propose in this work the use of crowdosurcing techniques to synchronize these videos, instead of an automatic one. Crowdsourcing [Howe 2006] in our scope refers to the use of the crowd as part of a computational problem that can be solved easily by a human than by a machine. The "easily" word can mean that the human approach: is cheaper, faster or can be done more efficiently by humans.

In video synchronization, we know that humans can fulfil all challenges presented by [Schweiger et al. 2013]. A person can identify if two videos are synchronized or not independently of occlusions, change of the background, camera motion or view point changes. The main challenge is how to

The work is supported by CAPES and FAPES.

use the human abilities to synchronize the videos, and permit that other persons can benefit from these contributions. So our tool uses the power of the crowd to synchronize live streaming videos and provide a form that other persons that want to watch those videos can receive both videos and synchronization info.

The remaining of this paper is presented as follows: section 2 explains our approach to synchronize videos; section 3 details our tool; and section 4 presents our final remarks.

Nowadays users don't need to rent fancy and expensive devices, neither depended on professionals to produce and share video, they can with their mobiles create User Generated Video (UGV). UGV is a kind of multimedia content created by heterogeneous users, shared online and without any explicit coordination mechanism. UGVs can be grouped around an event: a particular geographical space shared by a group of people at a particular period of time, such as protests, music festivals or sport games, resulting in awesome enahnced contents that can be reused in many ways. Moreover, each one of these videos reveals a unique point of view about what is happening, according to the user's identity and beliefs (in terms of ideology, team and group identification etc.) as well as user's context and preferences (in terms of positioning, device capabilities and limitations etc.).

In this scenario, it is impossible to ensure that all the UGV related to a same significant moment in a social event will be stable, have similar visual and aural quality or even if this moment was captured by the user. Then, automatic video synchronization techniques are not so effective. The synchronization of UVG about a particular topic or event can be announced as storytelling problem. Intuitively, this problem can also viewed as a synchronization problem, in which all of the related contents must be, firstly, positioned in a same global timeline and, in the sequence, arranged to produce a coherent narrative flowing.

Focused on this scenario, this paper introduces the idea of using crowdsourcing techniques for UGV synchronization, once the human processing is less affected by the heterogeneity of UGV content. Hence, we claim that the process of finding the synchronization points between UVG should involve crowd workers when automatic techniques are not effective. Also, the announced synchronization problem must consider many issues and in this paper, we focus on a particular issue: how to manage both crowd and video synchronization information and use them to rebuild the story of an event. In this sense, the paper introduces the Dynamic Alignment List (DAL), a data structure to assist the crowd contributions process as well as the generation of a coherent presentation of an event using UVG content.

We conducted three experiments to to investigate if the crowd would perform the proposed approach: the first used a crowd simulator to verify the DAL capability of managing videos and contributions, generating coherent presentations from user generated videos; the second experiment used a crowd to synchronize videos performing small tasks. The LiveSync Tool, developed to test if the crowd can be used to synchronize live streaming events, is detailed and shows how to use the structure to synchronize live strems. Last but not least, the focus of the paper is to show that the crowd can be used in scenarios where automatic techniques may find challenges, not to state that the use of the crowd is better than the automatic techniques.

2. CROWDSOURCING

Crowdsourcing is often a highly structured process from an organization, drawing on the creativity and intelligence of an online community in an open, but controlled, way [Brabham et al. 2014]. Crowdsourcing systems can be used in a variety of situations such as: Knowledge Discovery and Management; Distributed Human Intelligence Tasking Organization; Broadcast Search; Peer-Vetted Creative Production [Brabham et al. 2014]. The CrowdSync problem can be classified as a "Distributed Human Intelligent Tasking" problem, where an organization sends task related to information analysis. This kind of system deals with an information management problem: how the crowd will use information

contained in videos to generate synchronization points. A second characteristic for our problem is that we consider that the information is already available and it is unnecessary to locate it. We don't approach the problem of collecting the videos, here we consider only synchronizing them.

Synchonizing videos isn't the only way to use the crowd with videos. Several works use the crowd with other objectives on videos. Two main problems may be highlighted here: annotation and quality evaluation, but diverse works can be addressed.

2.1 Annotation

The paper Video Summarization via Crowdsourcing [Wu et al. 2011] uses the crowd to identify the main occurrences in a video and thus generating a summary. In Efficiently Scaling up Crowdsourced Video Annotations [Vondrick et al. 2013] the crowd is used to annotate multiple videos. Through an interface provided by the authors, users can annotate videos using graphic tools, classifying objects in a video. Besides the annotation, which uses crowdsourcing, the paper presents a complementary approach that helps in annotating videos. Authors use annotations from the crowd as an input to an algorithm that automatically finalizes annotations from a video.

Generating Annotations for How-to Videos Using Crowdsourcing [Nguyen et al. 2013] also uses the crowd to generate annotations, however, in this case within a specific context. Authors divide the annotation task in three steps: identifying the time when important events occur; name each event; and identify frames indicating instants before and after the event, helping in the How-To.

Crowdsourcing event detection in YouTube video [Steiner et al. 2011] focuses in generally identifying events in a YouTube video. It features processing in video playback time, to identify scene changes, and then allowing a viewer to annotate that part of the video. However, as stated by authors: "Regarded in isolation, neither of our video event analysis steps is new". Its contributions are in: (i) scalability through crowdsourcing, (ii) the nature of real time processing in a HTML5 client, and (iii) the combination of annotations for three different types of events (visual event, occurrence event and interest-based event).

A web-based video annotation system for crowdsourcing surveillance videos [Gadgil et al. 2014] presents a platform for annotating surveillance videos. A supervisor selects and assigns tasks to users. The paper, despite using the term crowdsourcing , uses the concept of outsourcing, once results from the crowd are not evaluated, neither combined to generate the result. Each individual has to watch the entire surveillance video and annotate it, falling back to the large tasks problem.

Tagging human activities in video by crowdsourcing [Nguyen-Dinh et al. 2013] uses the crowd to annotate scenes in a video, where each participant must annotate its start, end and details of its content to generate an annotation.

2.2 Quality

In Quantification of YouTube QoE via Crowdsourcing [Hoßfeld et al. 2011] the crowd is used when evaluating YouTube's quality of services (QoS). Each crowd individual watches videos from his home and rates his experience. An important point of the paper is the crowd filtering: in multiple tests executions, nearly 80% of them were removed from results as they were considered to fail the evaluation. For the elimination, these techniques were applied: Golden Standard Data; Consistency Tests; Content Quests; Mixed Answers; and Application Usage Monitoring. Video quality evaluation in the cloud [Keimel et al. 2012] also presents a quality evaluation, but this time an evaluation from the actual video.

A subjective evaluation using crowdsourcing of Adaptive Media Playout utilizing audio-visual content features [Rainer and Timmerer 2014] presents another paper that uses the crowd to evaluate

video quality. In this case, issues concerning adaptive video are evaluated. It is asked to participants to evaluate video quality, without knowing if video quality was modified or it remains the same.

2.3 Others

Crowdsourced Automatic Zoom and Scroll for Video Retargeting [Carlier et al. 2010] uses the crowd to identify focusing regions of a video. Viewers select a zoom area that focuses important content from the video, so when the video is watched in low resolution screens, only most important content is presented to users.

Introducing game elements in crowdsourced video captioning by non-experts, [Kacorri et al. 2014] presents the use of game elements (gamification) in a crowdsourcing platform. In this case, the problem that needs to be solved is generating captions for multiple videos. The most significant about this paper is the use of video segments. This allows users to perform small tasks to achieve a greater goal. Gamification aspects are important as an incentive to participation, instead of monetizing tasks.

3. VIDEO SYNCHRONIZATION

Besides studying the relation between crowds and videos segments, it is important to consider other video synchronization techniques, and that is what we now describe now in this section.

The audio analyses can be used to synchronize the video segments. Su *et al.* [Su et al. 2012] presents video synchronization using this approach. Fingerprints are generated for each audio, and a comparison between videos is performed. Bano *et al.* [Bano and Cavallaro 2015] also use audio as synchronization track using the chroma analysis from audio to group and synchronize audio from a same event.

There are also the approaches where video analyses is used instead of audio. Wang *et al.* [Wang et al. 2014], synchronizes videos in space and time, allowing the navigation between videos by resemblance and time. The synchronization works for multiple videos and different cameras. Other important works is described by Schweiger *et al.* [Schweiger et al. 2013] that a research for related papers in the area. It presents important results in automatic video synchronization, such as a technique that analyses differences between frames to find synchronization points. Furthermore, it presents the main challenges for automatic synchronization algorithms: wide baselines, camera motion, camera shaking, dynamic backgrounds and occlusions.

The use of human perception however, is not impaired by these. The human processing can overcame all these challenges and with or without the sound information. This is the principle that guided us to the proposed CrowdSync. Here we don't claim that the CrowdSync overcomes the automatic techniques, but with the crowd we find less limitations on what videos we can synchronize.

4. CROWDSOURCED VIDEO SYNCHRONIZATION

A traditional video presentation involves a single User Device that is able to decode and present this single content (the Main Content) originated from a unique source. In a non-traditional scenario [Huang et al. 2013], the presentation environment is composed of multiple user devices (TV, smart phones, tablets, etc.) able to present multiple contents delivered for multiple sources. One such scenario is an user that access a web video and is able to access other correlated videos with different angles, audio and complementary information.

In this situation, the user accesses a mashup of digital contents that may have no explicit synchronization defined to orchestrate the presentation. Mashups are applications generated by combining content, presentation or other applications functionalities from disparate sources. They aim to combine these sources to create useful new applications or services to users [Yu et al. 2008]. This combination

of services and contents however brings the following issue: how to synchronize these multiple contents for each user in its environment, since the contents are transmitted through different channels and from different sources that are not explicitly synchronized among them? To tackle this issue we use the crowd as part of our solution. They act as couplers, in other words, they are responsible for finding the synchronization points among related videos, allowing their synchronous presentation in a mashup video application.

The goal of video synchronization is to align a set of videos α in a common temporal line [Segundo and Santos 2015]. For this purpose, consider $\alpha 1$ and $\alpha 2$ as two continuous videos. They are considered to be synchronized when, in a given time T_k at the kn^{th} time instant of $\alpha 1$ and T_M at the mn^{th} time instant of $\alpha 2$, they both correspond to the same instant in global time when they were captured, which is an instance of continuous space-time. If they are not synchronized, there is a time offset (Δ) that added to the presentation of $\alpha 1$ or $\alpha 2$ will make them synchronous. The time offset between two videos V1 and V2 can be defined as $\Delta_{V1,V2} = b.V1 - b.V2$, where "b.V" is the starting time of video V in reference to a timeline of the related videos, and Δ is the time offset between them in this timeline. Finding this Δ is the task attributed to the crowd. They are responsible for analysing the videos, finding the correlated ones and setting the Δ that makes them synchronous.

Three different Crowd Synchronization scenarios are presented next: Chunk Synchronization, Frame Synchronization and Live Synchronization.

4.1 Chunk Synchronization

Following the crowdsourcing approach, we can't let each crowdworker analyse all the entire videos to find all synchronization points. This would require too much effort of each crowdworker. For the current version of our CrowdSync system, we split the videos in small chunks of 5s each. This way we make each task a lot easier to each crowd member, because for each task he needs only to compare if two chunks overlaps and if they do, what is the Δ that makes them synchronous.

Figure 1 shows this synchronization method. First each video A e B is mapped in chunks of 5s. A pair of chunks is sent to a crowdworker that evaluates if there is synchronization and what is the Δ if there is any. In the example (Figure 1), if the chunks [C_1A,C_1B] have no synchronization, we compare the next possibility [C_1A,C_2B] and so on until we find it or we compare all chunks. In the example the pair [C_1A,C_3B] contains a synchronization point. The crowdworker identifies it and discovers the time offset between them (Δ). Using the value of Δ and knowing which chunks where the ones where the synchronization point was found, we can synchronize the full videos [A,B]. In the example, the final difference between the two videos [A,B] is Δ plus two times the chunk size, because Δ was found in the relation between the first chunk of A and the third of B, a difference of two chunks.

When two chunks associated with two videos are synchronized, all the contents of these two videos will also be synchronized. This happens because as described in the introduction, we consider the videos as continuous. And if comparing all chunks from both videos, we can't find any synchronization point, those two probably have no synchronization point. We say probably, because there is a possibility that the crowd fails in its tasks. To reduce this possibility, there are measures that can be taken such as assessing the crowd for better contributions. We however do not discuss this issue in this paper, as our scope focus on the synchronization process only. Another important consideration here is that this approach can fall in the worst case when there is no synchronization point, and all chunks will be compared, and will fail. On the other hand, if there is an synchronization point, we can find it in the first comparison.

4.2 Frame Synchronization

The Frame synchronization is an enhancement proposal on top of the chunk model. Here, instead of using video chunks, each crowdworker receives the key-frames sequences from two videos that must be

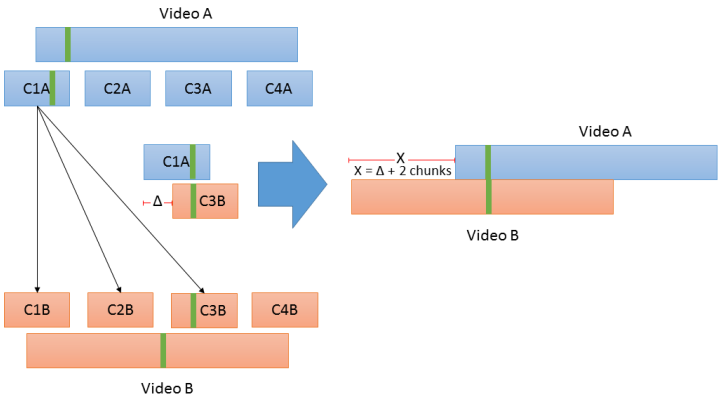

Fig. 1. Chunk Synchronization Method

synchronized. This way, each worker has access to the full video at once, enhancing the possibilities of finding a synchronization point. However, as now the worker interacts with frames, if a pair of frames is identified as a possible synchronization point, its precision is reduced, making necessary a second step on the task: watch the videos based on the information provided by the two frames (each frame has a timestamp from when it which part of the video it was removed, making possible to calculate Δ such as in the chunk based one).

Figure 2 shows the interface developed to this approach: on the top the user selects the probably aligned frames (or close ones) and plays the videos. If it is correct, he confirms the alignment, else he tries again or discard the synchronization.

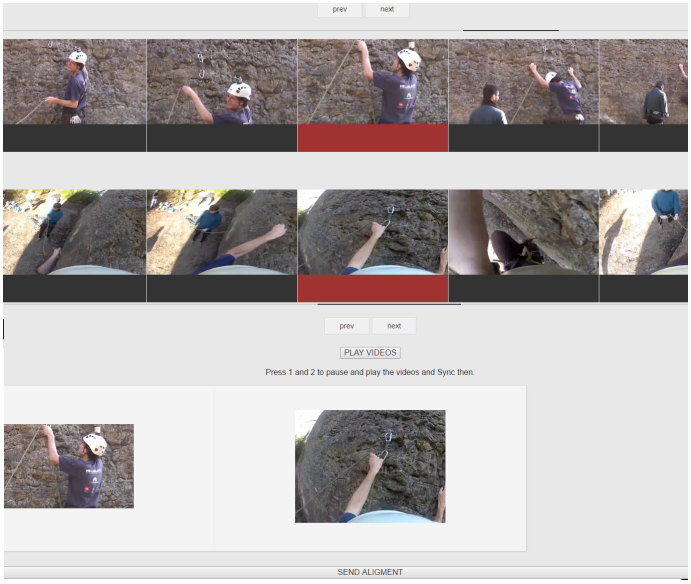

Fig. 2. Frame Synchronization Method

5. DYNAMIC ALIGNMENT LIST

To synchronize the video streams we use an synchronization technique presented in [Segundo and Santos 2015]: the Remote Temporal Couplers for aligning videos. It allows us to synchronize inde-

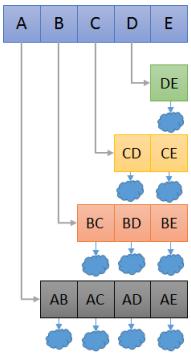

Fig. 3. DAL with 5 Assets (A,B,C,D,E)

pendent videos from multiple sources and to infer unknown relations from videos. We implemented the technique under our Dynamic Alignment List structure.

A DAL derivates from a matrix m x n, with m and n natural numbers, where m=n is the quantity of videos that can be synchronized at that moment. Each position of the matrix represents the relation of two videos (the coupler), in other words, the value necessary to align them, in our synchronization scenario. This value is calculated using: $\Delta_{i,j} = begin(id_{i,0}) - begin(id_{0,j}), para 0 < i, j \neq m$, where begin(x) is the time where an asset X begins its presentation. A $\Delta_{i,j} > 0$, implies that the asset in the column starts $\Delta_{i,j}$ before the asset in the line. Case $\Delta_{i,j} < 0$, the opposite happens and case $Delta_{i,j} = 0$, both starts at the same time.

As previously said, the list is derived from a matrix, presenting different aspects. Firstly, it does not presents all cells of a matrix. In the matrix, $\Delta_{i,j}$ is equal to $Delta_{j,i}$, so we don't need to store both relations, and with that, only the superior part of the relations are stored. Secondly, each "cell" of the list, presents multiple properties to help in the management of the crowd contributions. As each relation can contain multiple contributions, we need to calculate a mean of these contributions to set as the actual Δ. Lastly, each relation has a new dimension: a list with all contributions for that relation, that identifies the value of the contribution and the user that made it (if we want to identify who made it).

Figure 3, shows the representation of a DAL. There we have 5 assets (videos in our scope) labelled: A, B, C, D and E. Each asset represents a struct with the assets: URI, label, duration and a list of relations with the other assets. Following the asset A we find its relations with the other assets: AB, AC, AD and AE. We do not need to represent AA because $\Delta_{A,A}$ is always 0. Taking teh relations from B, we have: BC, BD and BE. Again, $\Delta_{B,B} = 0$. Thus, we do not represent BA, because we previously represented AB, and as said before, we can find BA by negating AB ($\Delta_{A,B} = -\Delta_{A,B}$). And this will happens to the other relations.

In the end of the DAL, we have the contributions for each asset (represented in the figure by a cloud due to figure dimensions). Each relation can receive multiple contributions, and are stored in the list. These contributions are processed (means) and generate the value of Δ for its relations. Fields such as confidence factor and number of contributions are used to know how robust those contributions are, and if its necessary more contributions to verify the synchronization.

Besides the synchronization method, there is the need to manage all the generated tasks: which pair of chunks will be sent to each user? From which videos? Where are the contributions stored? How can the contributions be validated? How to know that all videos are synchronized? What are the time offsets among the videos? Do I need to compare all videos?

Trying to answer these questions, we developed the concept of a Dynamic Alignment List. It is responsible for managing all contributions, checking the convergence of the contributions, distributing

the videos, inferring unknown values and storing Δ values for each pair of videos.

The DAL has two main features: (i) time offset and (ii) contribution managements. The time offset management deals with all videos relations manipulation while the contributions management takes care of the crowd management.

5.1 Structure

The DAL isn't just a list as the name may suggests, it is a data structure composed of vectors and lists that are used to assist the processing of contributions of crowd workers as well as to support the generation of presentations with the synchronized videos. It was designed in order to efficiently store all relevant contributions that represent a temporal relation between a pair of videos.

Its structure is based on a Upper Triangular Matrix, although, the DAL is constructed on demand, saving space because it doesn't have any empty slot on it, turning it into a dynamic structure. In Figure 4 it is shown an instance of a DAL with five videos and its relations.

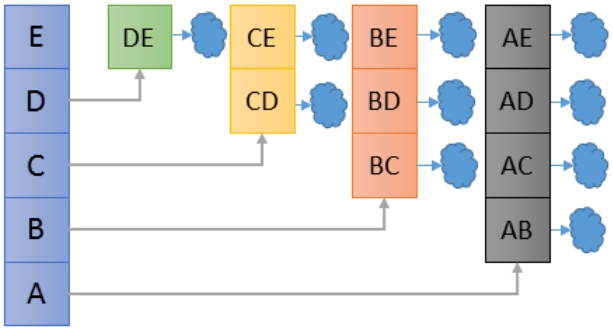

Fig. 4. DAL Structure

The DAL starts as an array that contains in its positions the videos that are going to be related (blue squares). Each video is an object that contains a reference to the media that it represents (URI) or other DAL (creating a hierarchy), a reference to the array of relations with the other videos and an attribute with the duration (σ) of the video.

These relations are cells into an array. Each position of this array is a different relation that represents the time offset between those two videos. Each position of the array stores the IDs for the pairs of videos that are being related, an attribute that contains the current Δ (time offset), an attribute that contains the degree of confidence in that relation and a list of contributions on that relation (each contribution on which is the Δ value). The degree of confidence is how precise the current Δ is believed to be true. Note that we don't need to store all direct relations (e.g. $\Delta_{A,H}$ and $\Delta_{H,A}$), we can store one direct relation and its complementary can be easily calculated ($\Delta_{A,H} = -\Delta_{H,A}$).

A contribution (in the blue cloud) in turn has the Δ proposed by each user for that relation and a reference for that user (the user profile may be used to rise the degree of confidence of a relation).

5.2 Time Offset Management

The first feature (and original purpose) of the DAL is the Time Offset Management. It is responsible for dealing with all aspects of the time offset relation among videos. These aspects are:

Δ Storage and Retrieval: a characteristic of the DAL is to allow the addition, update and retrieval of any Time Offset for any pair of videos. This gave us the starting point of using a Matrix

like structure that permits fast access to any cell of the matrix from a pair of coordinates. This way knowing the two videos which we desire to know or to update the Δ, we can quickly obtain this information. However we don't use a pure matrix for two main reasons: information redundancy, we don't need to store the AB and BA relations, as one is the opposite of the other ($\Delta_{i,j} = -\Delta_{j,i}$, eq. 1); and a matrix doesn't allow the storage of multiple contributions in the same cell (requisite for the crowdsourcing);

$$\begin{aligned} \Delta_{i,j} &= b.i - b.j \\ \Delta_{j,i} &= b.j - b.i \\ -\Delta_{j,i} &= b.i + b.j \\ \Delta_{i,j} &= -\Delta_{j,i} \end{aligned} \tag{1}$$

Δ Inference: known relations in the DAL allows developing inference methods that use transitivity through pairs of aligned videos to calculate relations between videos,in which the offset is still unknown. If we know the relation AB and AC from the structure we can infer BC.

$$\begin{aligned} \Delta_{B,C} &= b.B - b.C \\ \Delta_{B,A} = b.B - b.A \rightarrow b.B &= \Delta_{B,A} + b.A \\ \Delta_{A,C} = b.A - b.C \rightarrow b.A &= \Delta_{A,C} + b.C \\ \Delta_{B,C} = \Delta_{B,A} + b.A - (b.A - \Delta_{A,C}) &\rightarrow \\ \Delta_{B,C} &= \Delta_{B,A} + \Delta_{A,C} \end{aligned} \tag{2}$$

Presentation Generation: With the known and inferred time offsets, it is possible to create a presentation of the event that correlates all videos with temporal relations. This presentation can be the maximal one, where the event is presented using the first video until the end of the last one, also when videos with overlapping exists, the option to change camera is made available to users.

5.3 Contribution Management

The Contribution Management is a requisite in order to adequate the DAL for a crowdsourced scenario. It is necessary to receive, distribute and process the crowd contributions in order to find the correct Time Offsets. The aspects involved in the Contribution Management are:

Convergence: one of the principles of crowdsourcing is collecting the contribution of multiple members and based on these contributions finding the solution to a problem. In our case, the crowd watches the videos fragments and find synchronization points. The convergence is responsible for getting all theses contribution, and merging them in actual results.

The Convergence Level detects the tendency of the Crowd about the Relation of a pair of videos. This tendency is an indicative of the agreement on a Delta between the videos. According to the scenario, a Convergence Threshold that is used to determine if a Relation is converged, is defined. Each Relation have an attribute that stores its current convergence level. If this level reaches a Convergence Threshold the method returns True, otherwise it returns False.

Initially all Relations are created with Convergence Level 0, and consecutive similar contributions increase this value. If a tendency change in the contributions is detected, the Convergence Level is reset.

Video Selection: the convergence deals with the problem of receiving and processing the crowd contributions. However, we must correctly choose the videos to be evaluated by the crowd that results in a contribution. The selection of these videos is part of the contribution management.

This method intent is to select which pair of videos should have priority to receive contributions. It

consists of a sequence of two other methods: (i) Choose the next Video; (ii) Choose the next Relation. When this method returns NULL, it means that the DAL has been converged.

In order to increase possible inferences over the contributions, the method try to spread contributions over the timeline, proceeding a random selection among the Videos that were not converged nor marked as a Impossible relation. Once a Video is chosen the method select from its Relations array the Relation closer to converge.

6. LIVESYNC

Live synchronization includes the scenario where a viewer has access to an event that is live streamed by more than one Content Provider. These content providers are independent, so their videos do not have initial resources that allow their automatic synchronization to viewers, requiring a video analysis to generate synchronization points (Couplers). This synchronization fits as a problem that can be solved by using the power of the crowd. This occurs because videos are generated independently, without synchronization points and without previous description of what is about to be shown in screen, neither how can it be correlated to other videos. This way, human perception is used in real time to generate unknown synchronization points.

As example for this scenario, we can take a public manifestation. In the event, multiple people can take their cell phones and start streaming the event. In their house, other people can watch the videos. However, the multiple videos from different sources will be asynchronous. We need a way to synchronize these UGV. We use the crowd to achieve it. To this objective we group all videos in a MashUp application that connects to a Coupler Server that contains all synchronization data. Both synchronizing and playing the videos are made using this mashup, that can receive videos from multiple sources.

Live synchronization includes the scenario where a viewer has access to an event that is live streamed by more than one Content Provider. These content providers are independent, so their videos do not have initial resources that allow their automatic synchronization to viewers, requiring a video analysis to generate synchronization points (Couplers).

As example for this scenario, we can take a public manifestation. In the event, multiple people can take their cell phones and start streaming the event. In their house, other people can watch the videos. However, the multiple videos from different sources will be asynchronous. We need a way to synchronize these UGVs. We use the crowd to achieve it. To this objective we group all videos in a mashup application that connects to a Coupler Server that contains all synchronization data. Both synchronizing and playing the videos are made using this mashup, that can receive videos from multiple sources.

In a live presentation, it is assumed that content must be consumed right after its generation. It is of extreme importance that the synchronization method can be performed in playback time, to allow the integration of live content. However, not all viewers are required to be part of the crowd to achieve synchronization. If a synchronization made by a single member of the crowd is accepted as accurate, it can be transferred to the remaining viewers, this way each one will have his content locally synchronized.

One way of live synchronization can work is as follows: a person selects and synchronizes two videos with the help of a manipulation tool. This becomes a candidate synchronization point. Several people can do the same, and the results can be based on multiple synchronizations. Having these synchronization points defined, synchronization information can be sent to other viewers interested in watching those videos. As simple example, take two independent sources that are transmitting an event. A mashup system allows the user to watch both videos at the same time is his device. However the videos are asynchronous and the user notices that. He then access the option to synchronize

the videos. After he achieve a synchronous result, implicitly his contribution is sent to a server that will feed other users that choose to watch the same videos with the synchronization specification. If the user thinks the content is not synchronised yet, he can synchronize it himself and send another contribution. This tool was implemented and is presented on section 5.

6.1 Method

In a live presentation, it is assumed that content must be consumed right after its generation. It is of extreme importance that the synchronization method can be performed in playback time, to allow the integration of live content.

The way synchronization is performed in live video poses a single requirement: in a live situation, there is no need to analyse an entire video to find synchronization points, only an estimated stream delay must be considered. This occurs because the event is happening in real time (live), sources are also live and, therefore, the lack of synchronization during playback is caused by video streaming delays. In this specific case, synchronization through the crowd may be achieved by using tools that allow a user to manipulate time from videos: he can, for instance, keep pausing videos alternately , until he feels they are synchronized. This is a consequence of having a reduced search space, when limited to stream delays between videos.

However, not all viewers are required to be part of the crowd to achieve synchronization. If a synchronization made by a single member of the crowd is accepted as accurate, it can be transferred to the remaining viewers, this way each one will have his content locally synchronized.

One way of live synchronization can work is as follows: a person selects and synchronizes two videos with the help of a manipulation tool. This becomes a candidate synchronization point. Several people can do the same, and the results can be based on multiple synchronizations. Having these synchronization points defined, synchronization information can be sent to other viewers interested in watching those videos. As simple example, take two independent sources that are transmitting an event. A mash-up system allows the user to watch both videos at the same time is his device. However the videos are asynchronous and the user notes that. He then access the option to synchronize the videos. After he achieve a synchronous result, implicitly his contribution is sent to a server that will feed other users that choose to watch the same videos with the synchronization specification. If the user thinks the content is not synchronised yet, he can synchronize it himself and send another contribution.

6.2 LiveSync Tool

The main functionalities provided by our tool are:

Synchronized Live Video Player: The tool permits users to watch multiple videos synchronized. He selects from a list of sources the videos he wishes to watch and them they are synchronized using information provided by other users. If a pair of videos does not have any information about their synchronization, users are invited to contribute and synchronize the videos.

Synchronization Infering: In some cases that there is no direct information about the synchronization of two videos, the tool is able to infer the synchronization about them, based on the contributions of other videos. We use the transitive attribute of video synchronizations, where if we know AB and BC synchronization info (couplers), we can infer AC. To infer this value we travel trough the DAL, finding the unknown relations (for example, CE), and try to find a path of known relations where we can infer CE. Taking the DAL in Figure 1, we can infer CE if we know: AC and AE. This a two steps route, but we try all possible routes when inferring, in a way that we fill as much relations as possible.

Video Aggregation: Although the focus of the LiveSync tool is on synchronization, we allow users to add new stream sources to the application. He only needs to set the video source, and the video will be added to the DAL and list of videos. However, we don't do any filtering about the added video, this means that the user can add any video to the application, even ones that contains none relation with the other videos. In future versions we plan to add options where other users can mark the video as not related, and then remove them.

Multiple Video Platform Support: One keypoint of our tool is the use of other platforms as video sources. The videos that we play to users and that are synchronized, are not provided by us, but by other live video stream platforms. To be compatible with our tool, two requisites are required:

(1) Remote Player: we need that the platform allows embeddable players on third pages, allowing us to control the player with its basic functionalities such as: play, pause and stop;
(2) Uptime Support: a second and fundamental requisite is an API that allows us to retrieve the video Uptime. Video uptime is the time since the beginning of the video that is presented on the video player. This is fundamental to create and replicate the couplers generated in synchronization process.

Serverless Architecture: Serverless architectures refer to applications that significantly depend on third-party services and putting much of the application behavior and logic on the front end. Such architectures remove the need for the traditional server system sitting behind an application.

Multiplatform: LiveSync is a Web Based application designed and developed in compatibility with HTML5 standard to its front-end (Mashup Player) component. It allows our application to run on multiple browsers, operational systems and devices.

Active X Passive Contributions: Two branches of the LiveSync are currently on our repositories. They differ only in one aspect: who defines what videos are to be synchronized: the crowd or the application? The active version allow users to navigate freely through the videos, synchronizing them when they wish to. The focus of this branch is to allow users to contribute if they want to. On the other hand, in the Passive branch the server gives the crowd exactly what video they will synchronize. The focus here is to rapidly synchronize all videos, so the focus isnâĂŹt to make users watch the videos, but force them to synchronize all the base for other porpoises. The active branch is the focus here, but can easily converted to the passive one.

6.2.1 *Main Functionalities.* The main functionalities provided by our tool are:

Synchronized Live Video Player: The tool permits users to watch multiple videos synchronized. He selects from a list of sources the videos he wishes to watch and them they are synchronized using information provided by other users.

: If a pair of videos does not have any information about their synchronization, users are invited to contribute and synchronize the videos.

Synchronization Infering: In some cases that there is no direct information about the synchronization of two videos, the tool is able to infer the synchronization about them, based on the contributions of other videos. We use the transitive attribute of video synchronizations, where if we know AB and BC synchronization info (couplers), we can infer AC. To infer this value we travel trough the DAL, finding the unknown relations (for example, CE), and try to find a path of known relations where we can infer CE. Taking the DAL in Figure 3, we can infer CE if we know: AC and AE. This a two steps route, but we try all possible routes when inferring, in a way that we fill as much relations as possible.

Video Aggregation: Although the focus of the LiveSync tool is on synchronization, we allow users to add new stream sources to the application. He only needs to set the video source, and the video will be added to the DAL and list of videos. However, we don't do any filtering about the added video, this means that the user can add any video to the application, even ones that contains none relation with the other videos. In future versions we plan to add options where other users can mark the video as not related, and then remove them.

Multiple Platform Support: One keypoint of our tool is the use of other platforms as video sources. The videos that we play to users and that are synchronized, are not provided by us, but by other live video stream platforms. To be compatible with our tool, two requisites are required:

(1) Remote Player: we need that the platform allows embeddable players on third pages, allowing us to control the player with its basic functionalities such as: play, pause and stop;

(2) Uptime Support: a second and fundamental requisite is an API that allows us to retrieve the video Uptime. Video uptime is the time since the beginning of the video that is presented on the video player. This is fundamental to create and replicate the couplers generated in synchronization process.

Serverless Architecture: Serverless architectures refer to applications that significantly depend on third-party services and putting much of the application behavior and logic on the front end. Such architectures remove the need for the traditional server system sitting behind an application [Roberts 2016]. More of this characteristic will be addressed in the next topic: Architecture.

Multiplatform: LiveSync is a Web Based application designed and developed in compatibility with HTML5 standard to its front-end (MashUp Player) component. It allows our application to run on multiple browsers, operational systems and devices.

Active X Passive Contributions: Two branches of the LiveSync are currently on our repositories. They differ only in one aspect: who defines what videos are to be synchronized: the crowd or the application? The active version allow users to navigate freely through the videos, synchronizing them when they wish to. The focus of this branch is to allow users to contribute if they want to. On the other hand, in the Passive branch the server gives the crowd exactly what video they will synchronize. The focus here is to rapidly synchronize all videos, so the focus isn't to make users watch the videos, but force them to synchronize all the base for other porpoises. The active branch is the focus here, but can easily converted to the passive one.

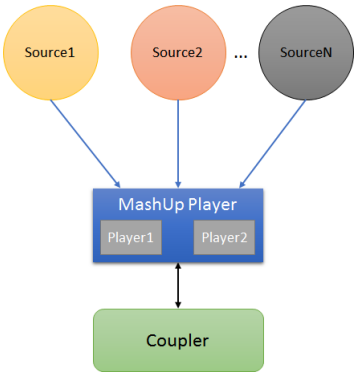

Fig. 5. Modelo simplificado do live-sync

6.2.2 *Architecture.* The LiveSync tool has three main components (Figure 5): the Content Providers (Video Sources), the Coupler and the Mashup Player.

The LiveSync tool has three main components (Figure 6): the Content Providers (Video Sources), the Coupler and the Mashup Player.

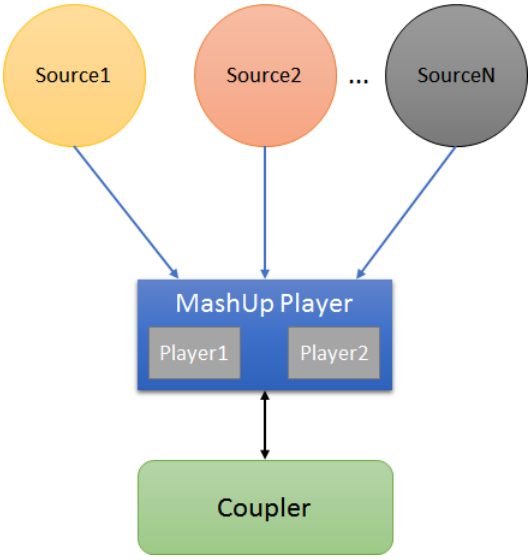

Fig. 6. LiveSync Model

6.2.3 *Content Providers.* Content Providers are third-parties videos streamers platforms. There are multiple Live Video Stream applications in the market, such as YouTube Live, LiveStream, TwitCast, Twitch and Ustream. One of our objective was to allow the use of different platforms as video sources, so we maximize the number of videos for an event and allow the use of already in market platforms.

A Content Provider needs two requisites to be compatible with LiveSync: a Remote Player and Uptime Support. As each stream platform uses their own protocols, we opt to use their embeddable players into a MashUp application. These players must allow us to play, pause and stop the video stream. The second requisite, Uptime Support, is necessary to find the couplers among the videos.

Uptime is the time passed since the beginning of the live stream until the video part being presented in the player at the moment of the call.

At this time, LiveSync supports two video stream platforms: the YouTube Live (https://goo.gl/DEM9eW) and WebSocket Stream (http://goo.gl/GYnGpg). YouTube live allows live stream from both desktop and mobile devices, making possible users to stream any event they wish with low effort: they only need to install the software and to have a YouTbe account. WebSocket Stream is an open source project that allows developers to implement live stream services with websockets and canvas and WebGl technologies. It works in any browser and was used for our first tests where we needed full control of the stream, something that YouTube Live doesn't support.

To add an video source form a Content Provider, it necessary only the video ID from YouTube Live or the video stream URI from the WebSocket Stream. These are added as assets to the LiveSync and can be accessed or synchronized.

Content Providers

Content Providers are third-parties videos streamers platforms. There are multiple Live Video Stream applications in the market, such as YouTube Live, LiveStream, TwitCast, Twitch and Ustream. One of our objective was to allow the use of different platforms as video sources, so we maximize the number of videos for an event and allow the use of already in market platforms. A Content Provider needs two requisites to be compatible with LiveSync: a Remote Player and Uptime Support. As each stream platform uses their own protocols, we opt to use their embeddable players into a MashUp application. These players must allow us to play, pause and stop the video stream. The second requisite, Uptime Support, is necessary to find the couplers among the videos. Uptime is the time passed since the beginning of the live stream until the video part being presented in the player at the moment of the call.

6.2.4 *Coupler.* The Coupler is responsible for storage, distribution and calculation of synchronization points among video streams from the Content Providers.

A coupler is composed of a DAL instance and Log files. This goes in direction of the Serverless Architecture. We wanted an architecture that needed low resources (another justification for using third party stream services) and easy deployment. All that is necessary do execute the coupler is a NODE.JS (https://nodejs.org/en/) server instance. This is possible because the Coupler is fully developed in JavaScript and compatible with the HTML5 standards. To deploy the Coupler, we use a Backend as a Service or "BaaS" platform, more specifically we use the Heroku (www.heroku.com) one, that permits free use of NODE.JS instances.

It stores synchronization information only during the duration of the event, so its stance is finished with the end of the videos and all data is lost. In the current scope, the sync info is only necessary during the event, after it, there is no need to store the information. For reasons of testing and using the filmed videos from YouTube we create log files that contains all contributions made by the crowd. If it is important to maintain all contributions and data for post analyses and further use, unstable version of the LiveSync is being configured to use a fully transactional database. We use a fully transactional database because we want to maintain track of all contributions made by the crowd, an important aspect in crowdsourcing and that is also supported by the DAL.

Other aspect of the Coupler is that it is responsible for the distribution of synchronization couplers. When a user chooses two videos, a message is sent from the MashUp to the coupler, containing the required relation. The coupler then answer with the required information. If the relations is unknown, it answer soliciting the user to synchronize and contribute with those two videos.

The last function of the Coupler, is to calculate the synchronization points among Videos. Each relation ($\Delta_{A,B}$) may contain several contributions, then it is necessary to calculate a value to that

relation based on the contributions. In the current version we calculate a Geometric Mean of the contributions to find an ideal value. This however may not be the best value, because the more accurate the sync is, the better the results are, so older contributions must have a lighter weight, something that does not happen in current version. The other calculation made by the coupler, is to infer unknown relations based on the afore mentioned transitive relation.

The communication between Coupler and MashUp is made through Websocket communication. The MashUp creates a WebSocket channel with the Coupler, and requests the sync information or sends contributions from the crowd. A simple protocol is used in JSON messages: act:value, data:object. The act field contains the action to be made and the data contains an object to complement the action. As example we have an act to send a new contribution ("contribution") that is complemented with a new relation that contains the assets involved, the value and an id to that contribution.

Coupler

The Coupler is responsible for storage, distribution and calculation of synchronization points among video streams from the Content Providers.

A coupler is composed of a DAL instance and Log files. This goes in direction of the Serverless Architecture. We wanted an architecture that needed low resources (another justification for using third party stream services) and easy deployment. All that is necessary do execute the coupler is a NODE.JS (https://nodejs.org/en/) server instance. This is possible because the Coupler is fully developed in JavaScript and compatible with the HTML5 standards. To deploy the Coupler, we use a Backend as a Service or âĂİBaaSâĂİ platform, more specifically we use the Heroku (www.heroku.com) one, that permits free use of NODE.JS instances.

It stores synchronization information only during the duration of the event, so its stance is finished with the end of the videos and all data is lost. In the current scope, the sync info is only necessary during the event, after it, there is no need to store the information. For reasons of testing and using the filmed videos from YouTube we create log files that contains all contributions made by the crowd. If it is important to maintain all contributions and data for post analyses and further use, unstable version of the LiveSync is being configured to use a fully transactional database. We use a fully transactional database because we want to maintain track of all contributions made by the crowd, an important aspect in crowdsourcing and that is also supported by the DAL.

6.2.5 *MashUp Player.* Mashups are applications generated by combining content, presentation or other applications functionalities from disparate sources. They aim to combine these sources to create useful new applications or services (the offer and consumption of data between two devices) to users [Yu et al. 2008]. In LiveSync we combine videos coming from different sources and platforms with the synchronization information from the coupler to reproduce a synchronous presentation of these videos.

The MashUp Player (Figure 3) is responsible for both presenting video synchronously and collecting the synchronization. Figure 7 represents the interface of the MashUp.

On the top we have all information necessary to the user. He can aggregate new videos using the ADD NEW VIDEO options, SYNCHRONIZE the videos if he thinks the videos are not synchronized or he can just select the videos he want to watch. When just playing two selected videos from the videos list, each video player creates an instance for the player that is compatible with that source (YouTube or WebSocket). It is invisible to the user where the video is coming from.

When the user adds a video, an input text is shown to him, and he can add the video URI (WebSocket) or video ID (YouTube). The page reloads and the new video is listed in the video list for everyone that connects to the application. When the video is added by the user, an message is sent to the Coupler, containing the action to add a new asset to the DAL, and the specification of it, such

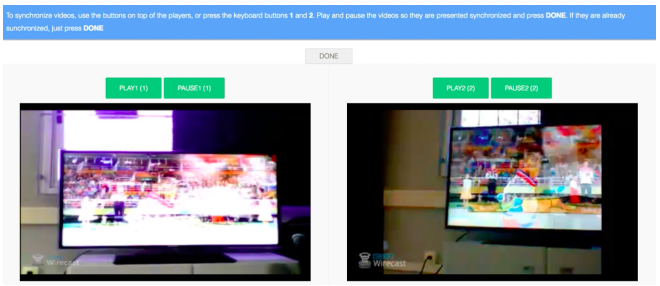

Fig. 7. Modelo simplificado do live-sync

as label and URI.

The last functionality of the MashUp is to synchronize the videos. When the user clicks on SYNCHRONIZE, a new mode of the application is revelled showing the synchronization tools. We use a Play 'n Pause approach to synchronize the videos. After the user thinks the videos are synchronized, clicking the DONE button, his contribution is sent to the Coupler and stored in the DAL for further processing of the relation.

Play 'n Pause: If two videos with a certain degree of similarity, are presented to an individual from the crowd, he can possibly notice that one video is ahead of the other. This way, he can pause the video that is ahead on time while the other remains playing, until reaching a point of synchronization. Then, the user can resume playback of the first video. This process can be repeated until an individual feels like both videos are being presented synchronously. On `https://goo.gl/HTzecL`, readers can see the play and pause technique being used to synchronize two live streams from from Two YouTube servers.

MashUp Player

Mashups are applications generated by combining content, presentation or other applications functionalities from disparate sources. They aim to combine these sources to create useful new applications or services (the offer and consumption of data between two devices) to users. In LiveSync we combine videos coming from different sources and platforms with the synchronization information from the coupler to reproduce a synchronous presentation of these videos.

The MashUp Player (Figure 8) is responsible for both presenting video synchronously and collecting the synchronization. Figure 8 represents the interface of the MashUp during a test: two cameras live streaming (content providers) a simulated television event to our mashup application.

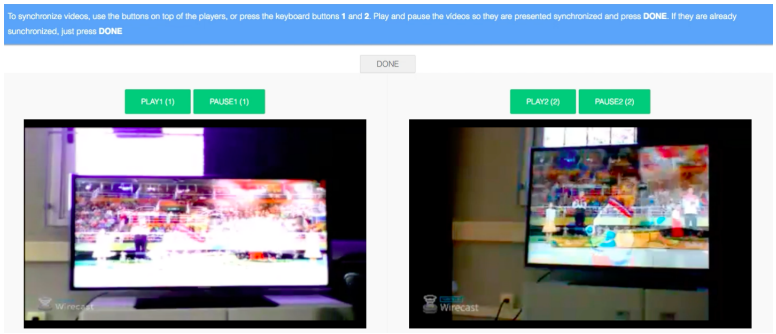

Fig. 8. Live Streams from Olympic Games Synchronized through two different cameras

On the top we have all information necessary to the user. He can aggregate new videos using the

ADD NEW VIDEO options, SYNCHRONIZE the videos if he thinks the videos are not synchronized or he can just select the videos he want to watch. When just playing two selected videos from the videos list, each video player creates an instance for the player that is compatible with that source (YouTube or WebSocket). It is invisible to the user where the video is coming from.

When the user adds a video, an input text is shown to him, and he can add the video URI (WebSocket) or video ID YouTube). The page reloads and the new video is listed in the video list for everyone that connects to the application. When the video is added by the user, an message is sent to the Coupler, containing the action to add a new asset to the DAL, and the specification of it, such as label and URI.

The last functionality of the MashUp is to synchronize the videos. When the user clicks on SYNCHRONIZE, a new mode of the application is revelled showing the synchronization tools. We use a Play âĂŹn Pause approach to synchronize the videos. After the user thinks the videos are synchronized, clicking the DONE button, his contribution is sent to the Coupler and stored in the DAL for further processing of the relation.

7. EXPERIMENTS

Aiming to verify the CrowdSync method and the data structure generated to implement the method, two different experiments were executed:

LiveSync Tool:. the LiveSync Tool implements all requisites for live synchronizing live UGV streams. Its objective is to allow the creation of mash applications of related video streams with client based solutions.

Crowd Simulated DAL:. the second experiment uses a simulated crowd to verify the DAL. The objective here is to verify if the DAL correctly stores, infers, converges and selects videos and relations. We simulate the crowd so we can make a controlled analysis in a set of videos with different crowd profiles.

UGV Dataset Synchronization:. in the third experiment, real users (crowd workers) synchronize a user generated video dataset using a developed platform with all characteristics described in this paper. Since the simulated experiment allows only to verify the correct functioning of the algorithms and structure, the focus here is to verify if humans using their perception can find the synchronization points in all related videos.

7.1 Crowd Simulated DAL

In order to validate the DAL it was designed a way to simulate a crowd, that allows us to run multiple experiments with large datasets in a viable time and without the possible biasing issues that can occur when using a human crowd. The solution found was to create the Crowd Simulator, a parameterized system that can simulate crowds with different levels of reliability, providing many contributions in short time.

To determine how the Crowd Simulator should work, we had to understand the behavior of the crowd. Yu et al.[Yu et al. 2012] classify the crowd in four categories: Hon workers: honest worker agents who return high quality HIT (human intelligence task) results randomly 90% of the time; MH workers: moderately honest worker agents who return high quality HIT results randomly 70% of the time; MM workers: moderately malicious worker agents who return high quality HIT results randomly 30% of the time; Mal workers: malicious worker agents who return high quality HIT results randomly 10% of the time. In our scenario we consider a high quality HIT as a positive synchronization point identification or correct negation.

Table I. Median for all 89 Videos, after 30 rolls of simulations for each Trustworthy degree

Reliability	Contributions	Relations	Error
100%	11161	1115	9%
90%	12101	1095	10%
80%	13199	1095	10%
70%	14475	1082	12%
60%	16077	1069	12%
50%	18015	1056	13%
40%	20500	1052	14%
30%	23904	1040	15%
20%	28404	1038	15%
10%	35121	1002	18%
0%	46356	975	21%

Per these definitions, the Crowd Simulator is settled with the percentage of honest workers in the simulated crowd, and we name this parameter as Crowd Trustworthy Degree. Each time the simulator is called, it determines the current worker reliability as so the chance of his contribution represents a high quality answer. As the objective of this experiment is validating de DAL, each high quality result is extracted from a Gold Standard Table that contains all relations between videos in dataset.

For this experiment we used the Climbing video dataset [Douze et al. 2016] that contains 89 videos captured from several different devices and users in order to register a climbing activity by distinct points of view. This climbing event has 4327.3 seconds long, and the videos starts in different offsets with durations varying between 18.96 and 1259.06 seconds. We made tests for 11 Crowd Trustworthy Degrees (from 100% to 0%) running 30 full simulations for each degree. Table 1 presents, for each Crowd Trustworthy Degree, the median of how many contributions were needed to converge the DAL (find all time offsets) after 30 roll of the simulation, the number of relations found, and the error rate.

As the dataset contains 89 videos, the goal was to find all 3916 relations for all pairs of videos in it. To reduce the false convergences, we used the convergence level three to determine that a value is assumed correct, this means that at least three sequential agreements are necessary to converge a value.

We also created a Gold Standard DAL built with 100% of correct answers from de Gold Standard Table (available with the dataset), allowing us to compare the DAL generated in each run with the Gold Standard DAL for accuracy and efficiency of the simulated crowd. Table I shows the results of the simulation. Each line corresponds to a test with different Trustworthy Degrees (reliability). Besides the reliability, each line presents the number of necessary contributions to converge, the total amount of possible relations (not all relations will present values, as there may be GAPs among videos) found among pairs of videos and the error degree.

Analyzing the result, we can see that the number of relations found donâĂŹt decrease significantly when the reliability of the crowd declines, also the percentage of the wrong relations found donâĂŹt raises dramatically. The major change among the different reliability degrees is on the number of necessary contributions to converge a DAL.

As can be seen in Figure 9, between Crowd Trustworthy Degree 100% and 50% the results decrease linearly, below 50% the result quality decline faster.

7.2 UGV Dataset Synchronization

Instead of discarding the human factor, this experiment aims at testing if the human factor is able to find and synchronize an User Generated Video dataset. The Climbing video dataset [Douze et al. 2016] presents multiple videos filmed by a group during a climbing activity. The automatic technique presented with the dataset presents some limitations making this dataset ideal to test the crowd

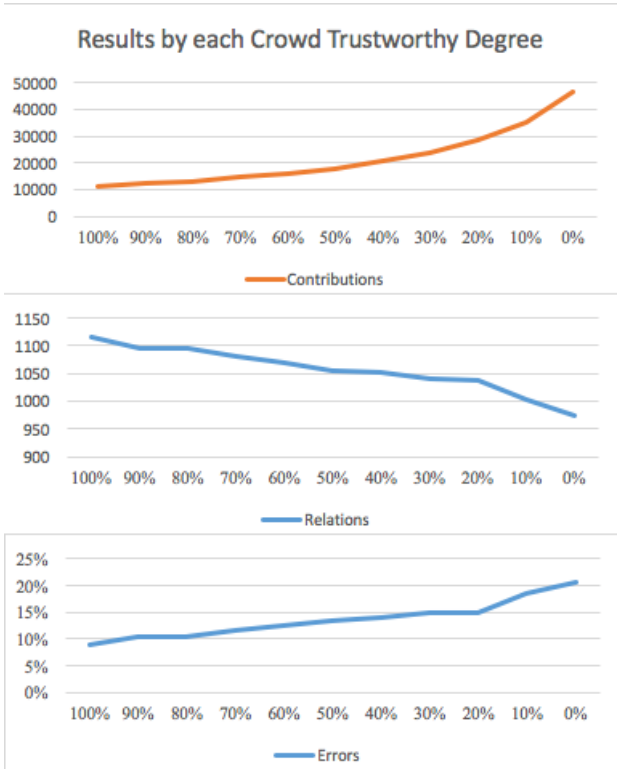

Fig. 9. evolution of results by each between Crowd Trustworthy Degree

approach, so we selected part of the dataset (9 videos) to our experiment. The automatic method could identify 23% of all pairs of videos that temporally overlap, the pairwise alignment score (PAS).

To execute our experiment, we developed the technique described in section 4.1, where the crowd member analyse video chunks (5s) pair by pair, identifying if there is relation and the precision of the relation if one exists. This was the chosen solution because requires less effort from the crowdworkers, and more detailed tests are necessary to evaluate the Frame Synchronization one, as a more complex task may compromise the worker activity.

From the known ground-truth we analysed the values resulted from the crowd to evaluate the alignment within a tolerance of 0.5s [Douze et al. 2016]. The comparison with the ground truth resulted in 88% (PAS) of correct relations among the videos. The nine videos generated 278 chunks of video (a total of 1390 seconds to be synchronized) demanding a total of 1051 contributions. Most of these (99%) contributions were negative ones, in other words, the crowd worker could not identify synchronization point. This indicates that most of the crowd effort is being done in discarding synchronization point, and not really finding them.

Figure 10 shows an instant where overlapping occurs in videos recorded using 6 different cameras are synchronously presented after the crowd contributions. The other three ones are not shown because in this instant no overlapping was indicated by the crowd. It is possible to notice how heterogeneous are the camera shots.

8. FINAL REMARKS

The LiveSync tool allows users to watch live video streams from multiple sources synchronized. If the videos are not synchronized, user can contribute and synchronize them- selves the videos and help

Fig. 10. Synchronized Video Matrix

building the synchronization, in a crowdsourced approach, believing that using human per- ception is better than using automatic approaches in this specific case.

Our tool however presents some limitations that shall in the futures be suppressed, such as the number of events that we can follow. Now, each instance of Coupler and MashUps can handle only one event, in other words, we canâĂŹt cover two independent live events at the same time with one instance, for that porpoise we need more than on instance of each service. Also, new stream services must be added to increase our compatibilities.

Github `https://github.com/rmcs87/liveSync` contains all code for LiveSync.

Crowds can be used in the most diverse situations: from designing a product to digitizing a word, going through most diverse scenarios, as discovering the structure of a protein. Exploring this variety of uses of the crowd this paper presented the possibility of using crowds in the video synchronization process. Using the crowd allows to address challenges that automatic processing techniques struggle to solve, like: moving cameras, constantly changing backgrounds, disappearing objects and others. A human can handle this problems without affecting his perception about a video

Nevertheless, using crowdsourcing techniques also introduces new problems to the process: which pair of videos will be sent to each user? Where are the contributions stored? How can the contributions be validated? How to know that all videos are synchronized? What are the time offsets among the videos? Do I need to compare all videos? As solution to these problems the DAL was described. An structure that can manage temporal relations and crowd contributions.

The first experiment showed the simpler where the crowd can be used: the live one. It is said simpler because the crowd has a smaller search area to find the videos alignment point. With the LiveSync this scenario can be solved and multiple mashups applications developed. The Second experiment showed us that the more reliable the crowd is, the less contributions we need and that 100% to 60% may generate similar results. Also we don't need the crowd to find all values: most of the relations we can infer from others. The third experiment showed us that it is possible to use the power of the crowd to synchronize video datasets, in the a hard dataset that presented challenge even to automatic solutions. From the second experiment we also learned that most contributions are to find that two chunks don't have a synchronization point. This is important and is the key point to the next steps in the development of our research, as we are developing new interfaces that helps the crowd members in identifying the synchronization points in entire videos at once, not only chunks. This will lead us in reducing the number of required contributions and achieve more accurate results.

All code involved in our research is opensource and is available in [Removed for Blind Review].

REFERENCES

Bano, S. and Cavallaro, A. Discovery and organization of multi-camera user-generated videos of the same event. *Information Sciences*, 2015.

Brabham, D. C., Ribisl, K. M., Kirchner, T. R., and Bernhardt, J. M. Crowdsourcing applications for public health. *American journal of Preventive Medicine*, 2014.

Carlier, A., Charvillat, V., Ooi, W. T., Grigoras, R., and Morin, G. Crowdsourced automatic zoom and scroll for video retargeting. In *Proceedings of the 18th ACM international conference on Multimedia*. ACM, pp. 201–210, 2010.

Douze, M., Revaud, J., Verbeek, J., Jégou, H., and Schmid, C. Circulant temporal encoding for video retrieval and temporal alignment. *International Journal of Computer Vision*, 2016.

Gadgil, N. J., Tahboub, K., Kirsh, D., and Delp, E. J. A web-based video annotation system for crowdsourcing surveillance videos. In *IS&T/SPIE Electronic Imaging*. International Society for Optics and Photonics, pp. 90270A–90270A, 2014.

Hossfeld, T., Seufert, M., Hirth, M., Zinner, T., Tran-Gia, P., and Schatz, R. Quantification of youtube qoe via crowdsourcing. In *Multimedia (ISM), 2011 IEEE International Symposium on*. IEEE, pp. 494–499, 2011.

Howe, J. The rise of crowdsourcing. *Wired magazine* 14 (6): 1–4, 2006.

Huang, Z., Nahrstedt, K., and Steinmetz, R. Evolution of temporal multimedia synchronization principles: A historical viewpoint. *ACM TOMM*, 2013.

Kacorri, H., Shinkawa, K., and Saito, S. Introducing game elements in crowdsourced video captioning by non-experts. In *Proceedings of the 11th Web for All Conference*. ACM, pp. 29, 2014.

Keimel, C., Habigt, J., Horch, C., and Diepold, K. Video quality evaluation in the cloud. In *International Packet Video Workshop)*. IEEE, 2012.

Nguyen, P., Kim, J., and Miller, R. C. Generating annotations for how-to videos using crowdsourcing. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*. ACM, pp. 835–840, 2013.

Nguyen-Dinh, L.-V., Waldburger, C., Roggen, D., and Tröster, G. Tagging human activities in video by crowdsourcing. In *Proceedings of the 3rd ACM conference on International conference on multimedia retrieval*. ACM, pp. 263–270, 2013.

Rainer, B. and Timmerer, C. A subjective evaluation using crowdsourcing of adaptive media playout utilizing audio-visual content features. In *IEEE Network Operations and Management Symposium*. IEEE, 2014.

Roberts, M. Serverless architectures, 2016.

Schweiger, F. et al. Fully automatic and frame-accurate video synchronization using bitrate sequences. *Multimedia, IEEE Transactions on* 15 (1): 1–14, 2013.

Segundo, R. and Santos, C. Remote temporal couplers for multiple content synchronization. In *Computer and Information Technology, 2015 IEEE International Conference on*. IEEE, pp. 532–539, 2015.

Steiner, T., Verborgh, R., Van de Walle, R., Hausenblas, M., and Vallés, J. G. Crowdsourcing event detection in youtube video. In *10th International Semantic Web Conference (ISWC 2011); 1st Workshop on Detection, Representation, and Exploitation of Events in the Semantic Web*. pp. 58–67, 2011.

Su, K., Naaman, M., Gurjar, A., Patel, M., and Ellis, D. P. Making a scene: alignment of complete sets of clips based on pairwise audio match. In *Proceedings of the 2nd ACM International Conference on Multimedia Retrieval*. ACM, pp. 26, 2012.

Vondrick, C., Patterson, D., and Ramanan, D. Efficiently scaling up crowdsourced video annotation. *International Journal of Computer Vision*, 2013.

Wang, O. et al. Videosnapping: Interactive synchronization of multiple videos. *ACM Transactions on Graphics (TOG)* 33 (4): 77, 2014.

Wu, S.-Y., Thawonmas, R., and Chen, K.-T. Video summarization via crowdsourcing. In *CHI'11 Extended Abstracts on Human Factors in Computing Systems*. ACM, pp. 1531–1536, 2011.

Yu, H., Shen, Z., Miao, C., and An, B. Challenges and opportunities for trust management in crowdsourcing. In *IEEE/WIC/ACM WI-IAT*. IEEE Computer Society, 2012.

Yu et al., J. Understanding mashup development. *Internet Computing, IEEE*, 2008.