Interactive and Automatic Navigation for 360° Video Playback

KYOUNGKOOK KANG, DGIST
SUNGHYUN CHO, DGIST

(a) 360° video (b) Normal Field-of-View (NFoV) video converted from (a) (c) Camera path for converting (a) into (b)

Fig. 1. Interactive and automatic navigation of 360° video*. Our system computes an optimal camera path that shows salient areas in a 360° video (a), and plays a NFoV video based on the path in an online manner. Users can interactively change the viewing direction while watching a video, and the system instantly updates the camera path according to user interaction. The y-axis in (c) is the horizontal angle in a 360° video. The solid blue line in (c) illustrates the camera path computed by our system, and the dotted line indicates the moment when the viewing direction is changed by user interaction. The solid blue line on the right of the dotted line is the updated path after user interaction. The original video is from the Pano2vid dataset [Su et al. 2016], which is originally from Youtube (fydc16RTfCo ©my360planet - Johannes Löffelmann). In the rest of the paper, we simply identify the name of the dataset, and the Youtube ID for each video.

A common way to view a 360° video on a 2D display is to crop and render a part of the video as a normal field-of-view (NFoV) video. While users can enjoy natural-looking NFoV videos using this approach, they need to constantly make manual adjustment of the viewing direction not to miss interesting events in the video. In this paper, we propose an interactive and automatic navigation system for comfortable 360° video playback. Our system finds a virtual camera path that shows the most salient areas through the video, generates a NFoV video based on the path, and plays it in an online manner. A user can interactively change the viewing direction while watching a video, and the system instantly updates the path reflecting the intention of the user. To enable online processing, we design our system consisting of an offline pre-processing step, and an online 360° video navigation step. The pre-processing step computes optical flow and saliency scores for an input video. Based on these, the online video navigation step computes an optimal camera path reflecting user interaction, and plays a NFoV video in an online manner. For improved user experience, we also introduce optical flow-based camera path planning, saliency-aware path update, and adaptive control of the temporal window size. Our experimental results including user studies show that our system provides more pleasant experience of watching 360° videos than existing approaches.

CCS Concepts: • **Computing methodologies** → **Computational photography**.

Additional Key Words and Phrases: 360° video, spherical panorama, user interaction, 360° video navigation

Authors' addresses: Kyoungkook Kang, DGIST, kkang@dgist.ac.kr; Sunghyun Cho, DGIST, scho@dgist.ac.kr.

0730-0301/2019/7-ART108 $15.00
https://doi.org/10.1145/3306346.3323046

ACM Reference Format:
Kyoungkook Kang and Sunghyun Cho. 2019. Interactive and Automatic Navigation for 360° Video Playback. *ACM Trans. Graph.* 38, 4, Article 108 (July 2019), 11 pages. https://doi.org/10.1145/3306346.3323046

1 INTRODUCTION

360° videos that record all directions at once are recently gaining popularity and getting widely available with the recent advent of virtual reality applications. Facebook and YouTube now support 360° videos so that users can easily watch 360° videos on mobile devices and computers. 360° cameras such as Samsung Gear 360, LG 360, and GoPro Fusion 360 have been rapidly emerging, enabling even casual users to easily create 360° videos. Moreover, commercially available virtual reality headsets such as Oculus Rift, Samsung Gear VR, HTC Vive, and PlayStation VR are getting popular too, accelerating production of new 360° video contents.

Because of the nature of 360° videos that record all directions at once, the most comfortable way for a viewer to watch such a video would be to wear a head-mounted display (HMD), and turn his/her head and body around to watch different directions. However, HMDs are not always available, and it is cumbersome to wear a HMD to watch a video every time. A more common scenario is to view a 360° video on a 2D display such as a computer screen or a smartphone.

One way to view a 360° video on a 2D display is to project the entire video using equirectangular projection so that a viewer can watch all directions at once. However, equirectangular projection introduces severe geometrical distortions (Fig. 1(a)). A more popular way widely used in many applications such as YouTube is to crop a part of a video, and render it as a normal field-of-view (NFoV) video. While watching a video, a viewer can manually change the viewing direction, e.g., by dragging a mouse or by orienting his/her phone. Unfortunately, it is uncomfortable either because a viewer needs to constantly adjust the viewing direction as the positions of

interesting events can constantly change. Moreover, other interesting events may exist in other directions, and it is easy to miss them unless a viewer carefully checks all directions constantly.

Recently, a few methods [Hu et al. 2017; Lai et al. 2017; Su and Grauman 2017b; Su et al. 2016] have been introduced to solve this problem and to allow viewers to more comfortably watch 360° videos. Given a 360° video, these methods automatically analyze its contents and find a virtual camera path that navigates the most interesting areas through the input video. A NFoV video is then rendered based on the obtained path so that a viewer can view interesting areas in the input video without any manual intervention. However, these methods produce a single NFoV video without any user interaction and users are not allowed to change the viewing direction when watching a video. Thus, other parts of the original 360° video except for the selected path are completely lost.

In this paper, we propose a novel interactive 360° video navigation system. Our system finds a high-quality camera path that shows salient events in an input 360° video, and displays a NFoV video based on the path. While watching a NFoV video, a user can change the viewing direction by dragging a mouse as in conventional 360° video players whenever he/she wants to. Then, the system instantly updates the path reflecting the user-specified direction.

To enable user interaction, our system is designed to consist of two steps: an offline pre-processing step, and an online interactive 360° video navigation step. In the pre-processing step, we estimate optical flow and saliency for a given 360° video. In the online navigation step, we split the input 360° video into temporal windows. Then, we find a camera path for one temporal window reflecting user interaction using one thread while playing a NFoV video of the previous window using another thread. For smooth user interaction, we introduce saliency-aware path update and adaptive control of the temporal window size.

Previous 360° video navigation methods [Hu et al. 2017; Su and Grauman 2017b; Su et al. 2016] often fail to track salient objects as they do not consider motions in the scene. For high-quality camera paths, our system finds a camera path reflecting the motions of objects and the camera using optical flow while maximizing regional saliency. Thanks to this, our method can effectively show dynamically moving salient objects and their surrounding context. The user study shows that our system without user interaction generates more pleasant NFoV videos than previous automatic methods, and our interactive system greatly improves user experience.

2 RELATED WORK

Automatic navigation of 360° video. The most relevant work to ours include [Hu et al. 2017; Lai et al. 2017; Su and Grauman 2017b; Su et al. 2016] that automatically generate NFoV videos from a 360° video. Su and Grauman [2016] divide the entire 360° video into multiple spatio-temporal segments, and estimate the capture-worthiness score for each segment using a learned classifier. Then, they find a smooth path that maximizes the capture-worthiness score by solving a dynamic programming problem. Finally, a NFoV video consisting of scenes worth watching is rendered based on the path. Su *et al.* [2017b] extended this method to incorporate zooming. However, they are limited to relatively static scenes and cannot handle dynamically moving objects or dynamic camera motion. It is because they use spatio-temporal segments of five seconds for capture-worthiness estimation, and a hard constraint for enforcing smoothness of the resulting camera path, restricting the camera path to move no more than 30 degrees for five seconds.

Hu *et al.* [2017] detect foreground objects at every frame of a 360° video, and use a selector network to find the main object among them. Then, a regressor network finds a smooth path that follows the main object. While this method can track dynamically moving objects more effectively than [Su and Grauman 2017b; Su et al. 2016], their networks sometimes produce temporally unstable results as temporal coherence is not explicitly enforced.

Lai *et al.* [2017] proposed a system to generate a NFoV hyperlapse video from a 360° video. The proposed system utilizes semantic segmentation labels, saliency, and the focus of expansion to find a camera path. The system also allows users to customize the result by choosing preferred semantic labels. However, the system is designed to generate a hyperlapse video instead of a normal-speed one, and the camera path is restricted by pre-defined semantic labels.

Besides all the differences between the aforementioned methods and ours, the most prominent one is that all the previous methods do not allow user interaction while viewing resulting NFoV videos. As a result, all the other parts of an original 360° video except for a small set of selected parts are completely lost in their resulting videos, and users cannot find out what is going on in the other parts. On the other hand, our system allows users to interactively change the viewing direction while watching a NFoV video so that they can navigate the video as they want to.

Interactive navigation of 360° video. Another approach closely related to our work is interactive navigation systems of 360° video. Lin *et al.* [2017b] recently proposed a visualization system that shows off-screen regions of interest as picture-in-picture previews. While watching a NFoV video, a user can select a preview to change the viewing direction to the selected region of interest. Pavel *et al.* [2017] presented a simple interactive system that re-orients a video to a region of interest when a user presses a button. In these systems, however, the viewing direction is fixed regardless of the motions of objects and the camera unless a user changes it manually. Therefore, a user still needs to constantly adjust the viewing direction to track an interesting event that changes its position. These interactive systems and ours are complementary to each other as they provide additional ways for user interaction. Moreover, as shown in [Lin et al. 2017a], users prefer different types of assistance for 360° video navigation depending on video contents. Thus, a combination of these methods and ours can further improve user experience as will be shown in Sec. 5.

360° images and videos. Due to the growing popularity of 360° images and videos, many different types of algorithms dedicated to 360° images and videos have recently been proposed. Jung *et al.* [2017] and Jeon *et al.* [2018] proposed image upright adjustment methods for 360° images. Kim *et al.* [2017] introduced a content-aware projection method that projects a part of a 360° video onto a 2D image plane minimizing distortions. Assens *et al.* [2017] predict a scan-path that describes how human visual attention moves on a 360° image. Su and Grauman [2017a] developed a specialized

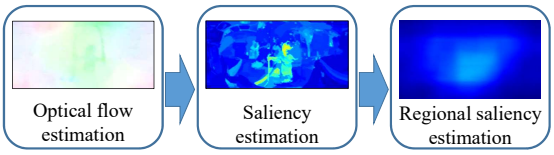


Fig. 2. Pipeline of the pre-processing step. The pre-processing step first computes optical flow from an input 360° video, and estimates saliency based on the optical flow. Finally, it integrates point-wise saliency scores to obtain regional saliency scores.

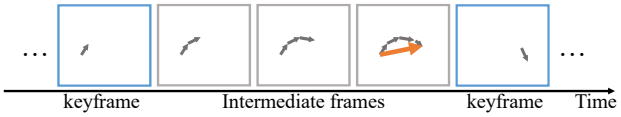


Fig. 3. Optical flow between consecutive key frames are computed by accumulating optical flow from all frames between them. Arrowed gray lines in each frame indicate optical flow vectors. The orange line is an obtained optical flow vector from a key frame to the next.

convolution operator for training convolutional neural networks on spherical panoramic images. Later, they also presented an approach to predict the rotation angle that yields the best compression rate for a 360° video [Su and Grauman 2018].

Video retargeting. Video retargeting has a similar goal to 360° video navigation, which is to fit a large 2D video into a smaller display possibly with a different aspect ratio. One approach to video retargeting is to detect and crop a salient part from an input 2D video. Liu *et al.* [2006] find a cropping window that moves in a restricted way, e.g., horizontal panning. Deselaers *et al.* [2008] find a cropping window from a 2D video that pans, scans, and zooms. Another approach to video retargeting is to apply content-aware warping to a 2D video instead of cropping. The most representative works in this direction include [Rubinstein et al. 2008, 2009; Wang et al. 2011, 2010; Wolf et al. 2007]. While video retargeting aims at a similar goal to ours, video retargeting methods do not need user interaction much because the sizes of original 2D videos and target displays in video retargeting are usually not much different.

Video stabilization. Video stabilization is a problem to produce a stable video from a shaky one as if the video was taken using a gimbal. Video stabilization bears a similarity to 360° video navigation, as it also finds a virtual camera path, and produces a cropped version of an input video. Video stabilization has been extensively studied due to its usefulness [Goldstein and Fattal 2012; Grundmann et al. 2011; Liu et al. 2009, 2011, 2013; Matsushita et al. 2006]. Jiang *et al.* [2014] and Liu *et al.* [2016] proposed online stabilization methods that produce a stable video at shooting time. Gleicher and Liu [2007; 2008] extended video stabilization and proposed re-cinematography methods to improve the camerawork of casual videos. For 360° videos, Kopf *et al.* [2016] introduced a method that corrects shaky rotational camera motion. However, despite extensive literature, most methods work fully automatically without user interaction. Bai *et al.* [2014] use user assistance for correcting motion estimation for high-quality video stabilization, but their goal to achieve using user interaction is different from ours.

3 360° VIDEO NAVIGATION

In this section, we explain our offline pre-processing step and camera path planning in the online navigation step. Our online navigation system with interactive path update will be discussed in Sec. 4.

3.1 Pre-processing

Fig. 2 shows the entire pipeline of the pre-processing step. Our system takes a spherical panorama video of size $W \times H$ as input. For efficient computation of optical flow and saliency scores, we first downsample the input panorama video to $W' \times H'$ where $W' = W/n$, $H' = H/n$, and n is a scaling factor. In our experiments, we set n so that $W' = 360$, which is the minimum width we found that optical flow and saliency can be reasonably estimated from. To compute accurate saliency scores and optical flow around the cut between the left and right boundaries of the input panorama video, the left and right boundaries are padded by 20 pixels in a circular fashion.

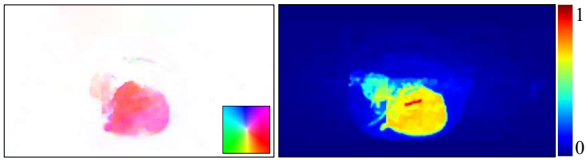
We then compute optical flow and video saliency using off-the-shelf methods for conventional 2D videos. Specifically, we use [Liu 2009] for optical flow. For saliency, we use[Zhou et al. 2014], which computes video saliency using optical flow as a motion cue. However, we note that any other methods can be used for computing optical flow and saliency scores. We also tested FlowNet 2.0 [Ilg et al. 2017], [Cheng et al. 2018], and capture-worthiness [Su et al. 2016] for optical flow and video saliency, but found that [Liu 2009] and [Zhou et al. 2014] produce the best results in our system. The resulting optical flow and saliency maps have the same spatial size as the downsampled input video. Saliency scores have values from 0 to 1, where 0 and 1 mean the least and the most salient, respectively.

For computational efficiency, we find an optimal camera path using only key frames, which are sampled every four frames, in the the online video navigation step. Thus, we compute saliency scores only for those key frames in the pre-processing step. We also compute optical flow between consecutive key frames by accumulating optical flow maps (Fig. 3). After computing optical flow and saliency maps, we crop the padded areas at the left and right boundaries.

We also crop the top and bottom of the optical flow and saliency maps by 10 pixels for computational efficiency and accurate camera path planning in the later step. Both the top and bottom parts of spherical panoramas have severe geometrical distortions that may harm the accuracy of optical flow and saliency. Moreover, as shown by Sitzmann *et al.* [?], they are likely to be less salient due to a bias of saliency to the equator. Thus, we simply exclude them when computing a camera path in the later step. We found that this works well in most cases, but we may sometimes need to consider the top and bottom parts. In that case, we may compute optical flow and saliency scores for those parts by projecting an input 360° video onto a cubemap as done in [Kopf 2016]. Fig. 4 shows examples of optical flow and saliency score maps. After cropping, we downsample both optical flow and saliency maps so that the resulting width is 180 pixels for efficient computation in the later step.

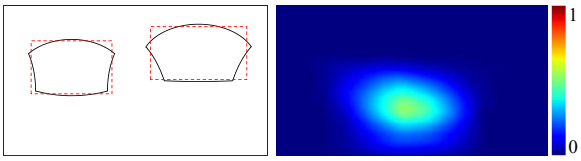
Finally, we compute the regional saliency scores of all possible NFoV video frames by integrating point-wise saliency scores. Let $s_t(p)$ is the saliency score of the t-th key frame at p where p is a 2D pixel coordinate in the downsampled spherical panorama video.

(a) Original frame



(b) Optical flow (c) Saliency score

Fig. 4. An example of optical flow and saliency score maps estimated in the pre-processing step. Optical flow vectors of different directions are visualized using different colors. The box on the bottom right in (b) shows how directions are mapped to colors. Original video from Sports-360 [Hu et al. 2017] (WwujLyXKNoo ©Roger Sanders).



(a) NFoV regions on a spherical panorama (b) Regional saliency computed from Fig. 4(c)

Fig. 5. NFoV regions are mapped to differently-shaped regions (black) on a spherical panorama (a). For efficient computation of regional saliency scores of different NFoV regions, we approximate the irregularly-shaped regions with rectangles (dotted red).

Then, the regional saliency $S_t(p)$ is defined as:

$$S_t(p) = \frac{1}{|R(p)|} \sum_{p' \in R(p)} s_t(p') \tag{1}$$

where $R(p)$ is a region centered at p that corresponds to a NFoV video frame. The shape and size of $R(p)$ is determined by the y-component of p and the field of view. In our system, we assume that the field of view is fixed. $|R(p)|$ is the number of pixels in $R(p)$. Due to the nonlinear mapping from a spherical panorama to a NFoV video frame, $R(p)$ has different shapes at different p as shown in Fig. 5(a). To efficiently compute $S_t(p)$, we adopt an approximation approach similar to [Su and Grauman 2017a] that approximates irregular regions on a spherical panorama with rectangles since $S_t(p)$ over a rectangular $R(p)$ can be efficiently computed using a summed-area table. Fig. 5(b) shows a regional saliency map corresponding to Fig. 4(c). We refer the readers to our supplementary material for more details about the approximation approach.

3.2 Camera Path Planning

In the online 360° video navigation step, we find an optimal camera path that passes through salient parts of the input 360° video and is smooth enough for a user to comfortably watch. Camera path planning consists of three steps: initial path planning, FoV-aware path planning, and path smoothing. The initial path planning step first computes a camera path that tracks the most salient object. The FoV-aware path planning step finds a path that is close to the initial path, but more effectively shows the surrounding context as well as

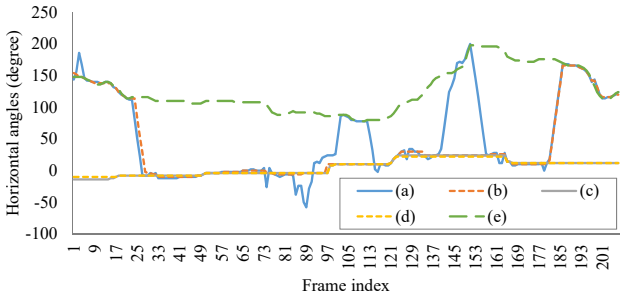


Fig. 6. Different camera paths computed by different methods. (a) Camera path computed by maximizing only saliency. (b)-(d) Camera paths obtained using the conventional smoothness term ((b) $\omega_o = 0.01$, (c) $\omega_o = 0.05$, (d) $\omega_o = 0.1$). A large ω_o for the conventional smoothness term makes the path stay at the same position not following salient objects, and a small ω_o makes the path jump between salient areas. On the other hand, our optical flow-based smoothness term encourages the path to follow salient objects more effectively and stably.

the most salient object. Finally, the path smoothing step computes a smooth camera path considering the velocity and acceleration of the virtual camera. As all the steps are performed during online video navigation, each step is designed to be lightweight.

3.2.1 Initial path planning. Given a set of key frames $F^{key} = \{f_1^{key}, \dots, f_T^{key}\}$, its corresponding saliency maps $\{s_1, \dots, s_T\}$, and optical flow maps $\{o_1, \dots, o_T\}$, we find an initial path $P = \{p_1, \dots, p_T\}$ where p_t is a 2D pixel coordinate in the downsampled spherical panorama video at the t-th key frame. We find P by minimizing the following energy function:

$$E(P) = \sum_{t=1}^{T} |1 - s_t(p_t)| + \omega_o \sum_{t=1}^{T-1} \|v(p_{t+1}, p_t) - o_t(p_t)\| \tag{2}$$

where $o_t(p_t)$ is the 2D optical flow vector at p_t from the t-th key frame to the $(t+1)$-th key frame. $v(p_{t+1}, p_t)$ is a 2D vector from p_t to p_{t+1} defined in a horizontally circular fashion, i.e., its x-component $v^x(p_{t+1}, p_t)$ is defined as:

$$v^x(p_{t+1}, p_t) = p_{t+1}^x - p_t^x + aW' \tag{3}$$

where p_t^x and p_{t+1}^x are the x-components of p_t and p_{t+1}, respectively, and a is either -1, 0, or +1 that gives the smallest $|v^x(p_{t+1}, p_t)|$. ω_o is a weight to balance the two terms. We set $\omega_o = 0.1$ by default in our implementation. $\|\cdot\|$ is an L^1 norm. The first term in the energy function is a saliency term that makes the path pass through the most salient parts of key frames, while the second term is a smoothness term that makes the path follow optical flow.

The smoothness term in Eq. (2) is a simple modification to a conventional smoothness term that encourages temporal change to be close to zero, which can be defined as:

$$\omega_o \sum_{t=1}^{T-1} \|v(p_{t+1}, p_t)\| \tag{4}$$

in our case. While our modification in Eq. (2) looks simple, it has a couple of noticeable advantages. First, it enables to track moving objects more effectively as optical flow reflects the motions of objects. Second, it helps avoid the path jumping back and forth between multiple salient objects. When multiple salient objects are present,

(a) Without FoV-aware path planning (b) With FoV-aware path planning

Fig. 7. FoV-aware path planning. While (a) shows the most salient object (the head of a woman) at the center, (b) looks more natural with other salient objects (a man on the left, and the body of the woman). Original video from Pano2vid (3d2PCTiNBAs ©Boonsri Dickinson Srinivasan).

maximizing saliency, or equivalently minimizing the first term in Eq. (2), may cause the camera path to jump between the salient objects. On the other hand, minimizing Eq. (4) may prevent the camera path from jumping between salient objects, but it also hinders the path from following fast-moving objects. Temporal smoothness based on optical flow can effectively solve this problem as it makes the path follow the optical flow of an object. Fig. 6 shows the effect of temporal smoothness based on optical flow.

We find an optimal solution of Eq. (2) using dynamic programming. Specifically, the energy $E_t(p_t)$ of an optimal path from the first key frame to the t-th key frame that ends at p_t can be recursively computed as:

$$\begin{aligned} E_t(p_t) &= |1 - s_t(p_t)| + E_{t-1}(p'_{t-1}) \\ &+ \omega_o \|v(p_t, p'_{t-1}) - o_{t-1}(p'_{t-1})\| \end{aligned} \tag{5}$$

where

$$p'_{t-1} = \underset{p' \in \mathcal{N}(p_t)}{\operatorname{argmin}} \left\{ E_{t-1}(p') + \omega_o \|v(p_t, p') - o_{t-1}(p')\| \right\}. \tag{6}$$

$\mathcal{N}(p_t)$ is a spatial neighborhood of p_t defined in a horizontally circular fashion. In our system, we set the size of spatial neighborhood to 31×31, which is large enough to track fast-moving objects on downsampled key frames. To solve dynamic programming, we compute $E_t(p_t)$ for all p_t's as we sequentially increase t from 1 to T. At the end of the process, the minimum $E_T(p_T)$ is the energy of an optimal path from the first key frame to the T-th key frame. The optimal path corresponding to the minimum $E_T(p_T)$ can be found by backtracking from the T-th key frame.

3.2.2 FoV-aware path planning. The initial path simply tracks the most salient events through the video. Thus, if we render a NFoV video based on the path, the video will always show the most salient object at its center regardless of its surrounding context. Unfortunately, as Fig. 7 shows, this may fail to yield an ideal result, especially when there is a large salient object, or are multiple salient objects close to each other. In such cases, a more ideal camera path would be one that shows not only the most salient object but also other ones.

To this end, we find a FoV-aware path $\tilde{P}$ that is close to P but reflects the field of view of a NFoV video. To obtain $\tilde{P}$, we minimize the following energy function for each key frame:

$$\tilde{E}(\tilde{p}_t) = |1 - S_t(\tilde{p}_t)| + \omega_p \|\tilde{p}_t - p_t\| \tag{7}$$

where $\tilde{p}_t \in \tilde{P}$ is the FoV-aware path at the t-th key frame. The first term on the right hand side promotes the FoV-aware path to cover more salient regions using regional saliency $S_t(\tilde{p}_t)$ and the second term encourages $\tilde{p}_t$ to be close to p_t. ω_p is a weight for the second term, which is set $\omega_p = 0.0001$ in our implementation. We do not include any terms to encourage temporal coherence in Eq. (7) as we have a separate path smoothing step, which will be discussed later. As Eq. (7) is defined independently to other frames, it can be minimized efficiently using exhaustive search. For computational efficiency, we also restrict $\tilde{p}_t$ to be within a 21×21-sized spatial neighborhood centered at p_t.

While we may directly find a FoV-aware path $\tilde{P}$ instead of P by replacing $s_t(p_t)$ by $S_t(p_t)$ in Eq. (2), we separately compute P and $\tilde{P}$ because they serve for different goals. A salient region in a video usually corresponds to an object, and P is computed to track the most salient object by maximizing the saliency at points. Then, $\tilde{P}$ is computed to show contextual information surrounding the main object by maximizing the regional saliency. In addition, we observed that direct computation of $\tilde{P}$ results in less satisfactory results because detecting and tracking a main object becomes ambiguous as $S_t(p_t)$ is a saliency score over a region, and the optical flow $o_t(p_t)$ at p_t no longer reflects the motion of the main object.

3.2.3 Path smoothing. Finally, in the path smoothing step, we compute a temporally smooth path $\hat{P} = \{\hat{p}_1, ...\hat{p}_T\}$ from the FoV-aware path $\tilde{P}$ by minimizing the following energy function:

$$\begin{aligned} \hat{E}(\hat{P}) &= \sum_{t=1}^{T} \|\hat{p}_t - \tilde{p}_t\|^2 + \omega_v \sum_{t=1}^{T-1} \|\hat{p}_{t+1} - \hat{p}_t\|^2 \\ &+ \omega_a \sum_{t=2}^{T-1} \|\hat{p}_{t+1} - 2\hat{p}_t + \hat{p}_{t-1}\|^2 \end{aligned} \tag{8}$$

where the first term on the right-hand side is a data term, and the second and third terms are smoothness terms based on velocity and acceleration, respectively. ω_v and ω_a are weights for the smoothness terms. We use $\omega_v = 200$ and $\omega_a = 2.0 \times 10^4$ in our implementation. The velocity-based smoothness term encourages the virtual camera to stay still, while the acceleration-based smoothness term encourages the camera to move at a constant speed like a panning or a dolly shot as discussed in [Grundmann et al. 2011].

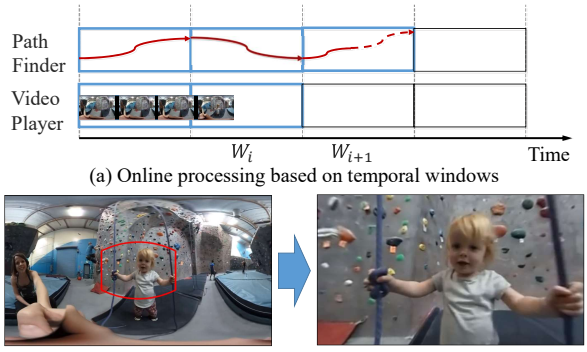
We may formulate a single energy function that unifies Eqs. (7) and (8) for a more optimal solution. However, the unified energy function leads to a discrete optimization problem involving saliency scores. We may solve the problem by dynamic programming but still need a huge search space and computation time due to the second order derivative-based temporal coherence term, which is not suitable for online processing.

In our system, before path smoothing, we upscale the FoV-aware path $\tilde{P}$ by multiplying $2n$ and linearly interpolate it for all intermediate frames first. Then, we perform the path smoothing step to obtain a smooth and properly-scaled path defined on all frames.

4 ONLINE 360° VIDEO NAVIGATION WITH INTERACTIVE PATH UPDATE

4.1 Online Video Navigation

To make our system run in an online manner, we split an input 360° video into disjoint temporal windows of N frames, which correspond to $N/4$ key frames. We denote the i-th temporal window by W_i. Our



(a) Online processing based on temporal windows

(b) Projection to a NFoV image

Fig. 8. Online processing. (a) Our system uses two threads: path finder and video player. The video player plays a NFoV video for the current temporal window W_i, while the path finder computes an optimal path for the next temporal window W_{i+1}. (b) Each spherical panorama video frame is projected as a NFoV image using the camera path as the projection center. The curved red box in the left image indicates the region that is projected as a NFoV frame. Original video from Pano2vid (pV_8ETmQ89w ©OurWorld360).

online step is implemented using two threads: path finder and video player. The path finder thread computes an optimal path for W_i while the video player thread plays a NFoV video of W_{i-1}. After playing a NFoV video of W_{i-1}, both threads proceed to the next windows (Fig. 8(a)). In our implementation, each temporal window has $N = 400$ frames, which correspond to 100 key frames.

The path finder thread finds a camera path for W_i based on the process described in Sec. 3.2. For temporal coherence between consecutive frames, both initial and smoothed paths should be coherent as they serve for different goals. To this end, we first modify the initial path planning as follows. We build a set of key frames F_i^{key} of W_i, and also include the last key frame of W_{i-1} as the first key frame in F_i^{key}. Then, we find an initial path for W_i by optimizing Eq. (2) with an additional constraint $p_{i,1} = p_{(i-1),T}$ where $p_{i,1}$ is the first camera position of the initial camera path of W_i and $p_{(i-1),T}$ is the last camera position of the initial camera path of W_{i-1}. This hard constraint can be easily implemented in dynamic programming by setting $E_1(p, T)$ in Eqs. (5) and (6) as follows:

$$E_1(p,T) = \begin{cases} 0 & \text{for } p = p_{(i-1),T} \\ \infty & \text{for other } p\text{'s} \end{cases} . \tag{9}$$

For coherent path smoothing, we introduce an additional hard constraint $\hat{p}_{i,1} = \hat{p}_{(i-1),T}$ to Eq. (8). Optimizing Eq. (8) with this equality constraint is a simple quadratic programming problem, which can be easily solved using Lagrange multiplier method [Wright and Nocedal 2006].

The video player thread renders NFoV video frames based on the computed camera path. Each frame in the input spherical panorama video is projected as a NFoV video frame using equirectangular projection. The camera path at the current frame is used as the projection center (Fig. 8(b)).

4.2 Interactive Path Update

For interactive path update, we designed a simple graphical user interface (GUI) as shown in Fig. 9, which looks similar to conventional video player apps. A user can watch a NFoV video generated from an input 360° video using our GUI. To change the viewing direction, a user can simply drag a mouse on the video player window. Then, our system updates the camera path reflecting the user input. Specifically, the system re-defines temporal windows from the current frame. Then, for each temporal window, a new camera path is computed according to the viewing direction specified by the user in an online manner.

Fig. 9. Graphical user interface showing a NFoV video with a progress bar on the bottom. A user can drag a mouse on the NFoV video to change the viewing direction.

To provide responsive path update with no latency, we adaptively change the size of temporal windows. Using a long temporal window can produce a more optimal path for a long period of time, but requires a long computation time for the camera path planning. On the contrary, a short temporal window can reduce the computation time, but may result in a less optimal path that does not follow salient objects. Fortunately, as will be discussed later, our system is designed to track the user-specified point instead of salient areas for a moment after a user changes the viewing direction, then gradually go back to track salient areas. Thus, even a very short temporal window does not degrade the quality of the path much. Therefore, we use short temporal windows when a user changes the viewing direction, and gradually increase the temporal window size.

Specifically, we set the frames from the current frame to the nearest key frame after τ seconds as the first temporal window. τ is used to give a short delay before resuming automatic navigation. We found that users often change the viewing direction by dragging the mouse several times in succession rather than once, and they feel uncomfortable if the viewing direction changes between mouse dragging in a preliminary experiment. To prevent the change of the viewing direction between successive mouse dragging, we set $\tau = 0.5$, which leads to the first temporal window of 15 to 18 frames.

For the first temporal window, we simply let the camera track the optical flow at the current position regardless of saliency. Thus the video player thread simply plays the first temporal window, and the path finder thread computes a camera path for the next temporal window. We set the size of the second temporal window to five key frames and double the size for each temporal window until it reaches at 100. Fig. 10 illustrates our interactive path update.

A camera path reflecting the user intention is obtained as follows. Let us first denote the pixel coordinate of the center point in the user-specified viewing direction by q. We assume that q is the point where the user is interested to watch. For interactive path update, we consider the following criteria:

(1) If q is in a region with high saliency, the updated path should follow the most salient object near the point.

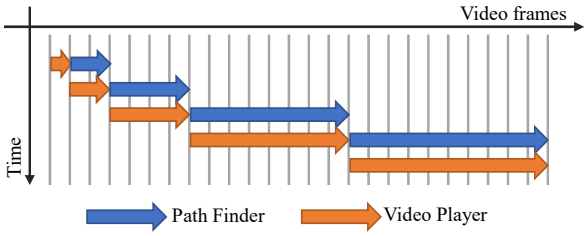


Fig. 10. Illustration of how interactive path update works. The path finder thread finds an optimal path for the next temporal window while the video player thread plays a NFoV video for the current temporal window. A small temporal window is used right after user interaction occurs to reduce the latency, and the window size gradually increases.

(2) Even if q is in a region with low saliency, we assume that the user still wants to see that direction. Thus, the user-specified direction should be shown for a certain amount of time.

To satisfy these criteria, we modify Eq. (2) as:

$$E(P) = \sum_{t=t_0}^{t_0+T} (1-\omega_t)|1-s_t(p_t)| + \sum_{t=t_0}^{t_0+T} \omega_t \|p_t - q_t\| + \omega_o \sum_{t=t_0}^{t_0+T-1} \|v(p_{t+1}, p_t) - o_t(p_t)\| \quad (10)$$

where t_0 is the index of the first key frame in the current temporal window, and q_t in the second term on the right hand side is the pixel coordinate corresponding to q on the t-th key frame, which is computed by accumulating optical flow. For the second temporal window where we resume automatic navigation, we use an additional hard constraint $p_{t_0} = q_{t_0}$.

ω_t is a weight that is initially 1 and decays to 0 as time goes. Specifically, we define ω_t as follows:

$$\omega_t = \exp\left(-\frac{|t-u|^2}{2\sigma^2}\right) \quad (11)$$

where u is the index of the key frame where we resume automatic navigation, and σ is a parameter that controls the decaying speed, which is determined according to saliency. If the user-specified point has high saliency, we set σ to be small so that ω_t quickly decays to 0 and the path follows a salient object satisfying the criterion (1). On the other hand, if the user-specified point has low saliency, we set σ to be large so that the path stays at the user-specified point for a longer period of time satisfying the criterion (2). We define σ as:

$$\sigma = \max\{\alpha(1 - s_u(q)), \epsilon\} \quad (12)$$

where α is a parameter to control the decaying speed of ω_t. We set $\alpha = 10$ in our experiments. ϵ is a small constant to prevent division by zero in Eq. (11). We use $\epsilon = 1e-4$ in our implementation.

5 RESULTS AND COMPARISONS

We implemented the pre-processing and online video navigation steps using Matlab and C++, respectively. We used the executables of the authors for optical flow [Liu 2009] and saliency estimation [Zhou et al. 2014], both of which are also implemented using Matlab and C++, and available on their websites[1]. We used the default parameter values presented in Secs. 3 and 4 for all the experiments as we empirically found that they performed well in most cases. Table 1 shows computation times for each step of our method, which are measured on a PC with an Intel Core-i7 3.7GHz CPU and 32GB RAM. The pre-processing step requires a long computation time mostly due to optical flow and saliency estimation. On the contrary, optimal path computation is fast enough to enable online path finding. The camera path planning takes 5.8 seconds for 100 key frames that correspond to about 13 seconds of a video of 30 frames per second. The camera path planning takes 0.3 seconds for 5 key frames enabling instant update of the camera path after user interaction. We note that our implementation is not optimized, and computation time can be further reduced by code optimization and adopting more efficient optical flow and saliency estimation methods.

Table 1. Computation time for each step. 100 key frames correspond to 13 seconds of a video of 30 frames per second.

Step	Time
Pre-processing of a 1 min. video	
Total	177 min.
Optical flow	10 min.
Saliency	166 min.
Regional saliency	0.37 sec.
Online 360° video navigation (100 key frames)	
Initial path	3.58 sec.
NFoV-aware path	2.14 sec.
Path smoothing	0.04 sec.

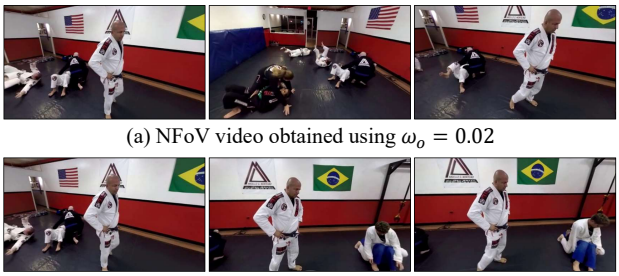
(a) Result using the conventional smoothness term

(b) Our result

Fig. 11. Example of multiple salient objects. Video frames are shown from left to right in the temporal order. The input video has multiple soccer players running on the field. While the result obtained using the conventional smoothness term fails to track the player in a red jersey, our result successfully tracks the player. Original video from Pano2vid (lvH89OkkKQ8 ©LOSC).

We tested our system using various videos including datasets of [Su et al. 2016], [Hu et al. 2017], and Youtube videos. We refer the readers to our supplementary material for all the resulting videos and comparisons shown in this section as well as additional examples and user study results. Fig. 11 shows our result on a 360° video with multiple fast-moving salient objects: soccer players running on the field. Both results are obtained without any user interaction. The first row shows our result using the smoothness term based on optical flow. The second row shows a result using the conventional smoothness term (Eq. (4)). To show the effect of the smoothness term more clearly, we did not use FoV-aware path planning in this

[1] https://people.csail.mit.edu/celiu/OpticalFlow/, https://github.com/zhfe99/sal



(a) NFoV video obtained using $\omega_o = 0.02$

(b) NFoV video obtained using $\omega_o = 0.1$

Fig. 12. Effect of different ω_o's. Large ω_o results in a stable path, while small ω_o makes the path able to jump between different events promptly. Original video from Pano2vid (70vmG1G7q0I ©Gotcha VR).

(a) Deep 360 Pilot [Hu et al. 2017]

(b) Our result

Fig. 13. Comparison with Deep 360 Pilot [Hu et al. 2017]. Video frames are shown from left to right in the temporal order. Hu *et al.*'s result shows abrupt jumping between two people, while our result does not.

example. Both videos initially track the player in a red jersey on the left. Our result keeps tracking the player until the end of the scene successfully thanks to the smoothness term reflecting optical flow, but the result with the conventional smoothness term fails to track the player and moves to another area. Fig. 12 shows the effect of ω_o that controls the balance between the saliency and temporal smoothness terms in Eq. (2). As shown in the figure, small ω_o allows the path to jump between different events more promptly, while large ω_o makes the path more stable.

We compare our system with previous automatic methods of Su *et al.* [2016] and Hu *et al.* [2017], which we refer to as AutoCam and Deep 360 Pilot, respectively, in the rest of this section. Fig. 13 shows a comparison between Deep 360 Pilot [Hu et al. 2017] and ours. Our result is obtained without any user interaction. Deep 360 Pilot finds a main object to follow using recurrent neural networks (RNNs). Even though the trained RNNs utilize multiple frames, they do not consider the motions of objects when choosing the main object. As a result, the result of Deep 360 Pilot shows one person, jumps to another, and then jumps back to the original person in a short period of time. On the contrary, our result shows a more stable camera path without jumping between different people, as we explicitly consider the motions of objects using optical flow.

(a) AutoCam [Su et al. 2016]

(b) Our result

Fig. 14. Comparison with AutoCam [Su et al. 2016]. Video frames are shown from left to right in the temporal order. The result of AutoCam fails to track a bike moving dynamically, and keeps showing the same direction, while our result tracks the bike more responsively. Original video from Sports-360 (PZwWP_oqB68 ©WOOZY BMX VIDEOS).

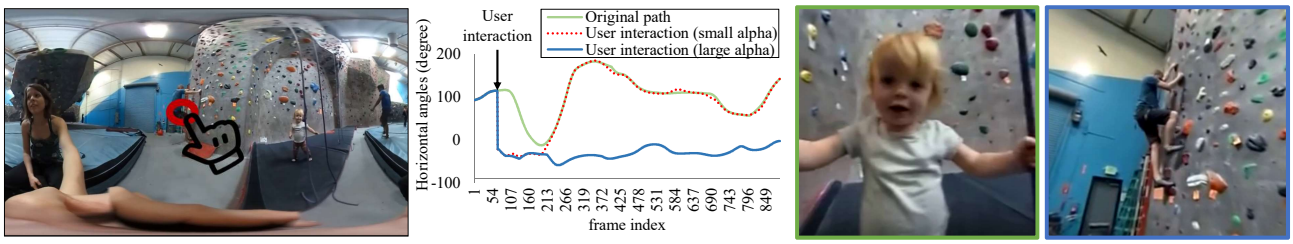
Fig. 14 shows a comparison between AutoCam [Su et al. 2016] and ours. The result of AutoCam was produced by Hu *et al.* [2017]'s implementation of AutoCam. AutoCam restricts the camera path to move less than 30 degrees for five seconds, as computing capture-worthiness scores requires a relatively long video sequence. As a result, its result fails to track a bike quickly crossing the scene, and keeps showing the same direction. In contrast, our result tracks the bike more responsively thanks to our effective path planning.

Finally, Fig. 15 shows an example of interactive path update. The green curve shows the camera path computed by our system without user interaction. The path follows a kid in the video as the kid is the most salient object. The black arrowed line indicates the moment when the viewing direction changes towards a man climbing a wall by user interaction. The solid blue and dotted red curves show updated camera paths by user interaction. Each curve is computed using different values for α that controls how long the path is affected by user interaction. For the blue curve, we use $\alpha = 30$ to make the path to stay at the position specified by the user for a longer period of time. For the red curve, we use $\alpha = 5$ to enable the path to more quickly move to more salient objects. At the beginning, both the blue and red curves track the kid exactly same as the green curve. After the user changes the viewing direction towards the man climbing the wall, the blue curve starts to track the man instead of the kid. The red curve also tracks the man for a while, then it goes back to the kid as the kid is more salient than the man.

5.1 User Study

5.1.1 Comparison with fully automatic methods. We conducted user studies to evaluate the performance of our system. To evaluate our system as a fully automatic 360° video navigation system, we compare ours with previous fully automatic methods: AutoCam [Su and Grauman 2017b; Su et al. 2016], Deep 360 Pilot [Hu et al. 2017], and 360 Hyperlapse [Lai et al. 2017].

For comparison with AutoCam [Su and Grauman 2017b; Su et al. 2016] and Deep 360 Pilot [Hu et al. 2017], we tried to use all possible videos provided by the previous works except for the videos whose input videos are no longer available on Youtube. To compare with



(a) Change of the viewing direction (b) Optimal camera paths (c) Video content in the original path (d) Video content in the user-specified direction

Fig. 15. Example of user interaction. The red circle in (a) indicates the viewing direction specified by user interaction. The green curve in (b) shows the original path when there is no user interaction. Red and blue curves in (b) show updated camera paths reflecting the user input with different α's.

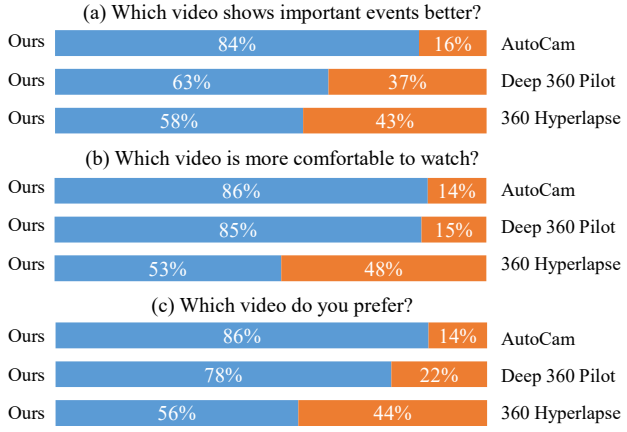


Fig. 16. User study result on comparison with other automatic methods.

AutoCam [Su and Grauman 2017b; Su et al. 2016], we used seven 360° videos, three of which are from the Sports-360 dataset of [Hu et al. 2017], and the other four are from [Su and Grauman 2017b]. Note that [Su and Grauman 2017b] is an extension of [Su et al. 2016] with zooming, so their results have similar tendencies. Thus, we group them together in our user study. For the results of AutoCam on the Sports-360 dataset, we used the results presented on the project website[2] of [Hu et al. 2017], which were generated using Hu *et al.*'s implementation of [Su et al. 2016]. For the other results of AutoCam, we used the results presented on the project website[3] of [Su and Grauman 2017b]. To compare with Deep 360 Pilot [Hu et al. 2017], we used nine 360° videos from [Su et al. 2016] and [Hu et al. 2017]. For the results of Deep 360 Pilot, we used the authors' results presented on their project website. As we use the videos downloaded from the project websites, some videos are of low resolution with compression artifacts. For fair comparison, we degraded our results as well by downsampling and upsampling.

To compare with 360 Hyperlapse [Lai et al. 2017], we modified our system to produce a hyperlapse video. Specifically, we first apply 360° video stabilization as done in [Lai et al. 2017], and then regularly sample input video frames to shorten the running time of the video. To estimate the optical flow between the sampled frames, we first estimate the optical flow between the consecutive input frames, then accumulate them as described in Sec. 3.1. Then, we perform the remaining process as described in Sec. 3. We used four representative videos (quickly moving forward, gliding, multiple salient objects, static camera motion) out of 10 video from [Su et al. 2016], which are used in Hyperlapse [Lai et al. 2017]. For the results of 360 Hyperlapse, we used the results of the authors presented in their project website[4].

For the user study, we recruited 20 participants. All the participants are graduate students majoring in either electrical engineering or computer science, but not related to computer graphics. Each participant was shown 20 pairs of videos. We first showed the input 360° video for each pair. Then, we showed the pair side-by-side. We randomly changed the order (left/right) for every pair, and did not inform the participants which methods were used to generate the videos. For each pair, we asked three questions to the participants: (a) which video shows important events better? (b) which video is more comfortable to watch? and (c) which video do you prefer?

Fig. 16 summarizes our user study result. More detailed analysis on the user study can be found in the supplementary material. Against AutoCam [Su and Grauman 2017b; Su et al. 2016] and Deep 360 Pilot [Hu et al. 2017], our method is preferred in all three questions by large margins, showing that our method is able to show important events in a more effective and comfortable way. In the comparison with AutoCam, most participants commented that our results show more information than AutoCam. There was also a negative feedback from one participant saying that it felt dizzy because the camera moved too quickly. In the comparison with Deep 360 Pilot, most participants commented that our results are less shaky and more comfortable to watch.

Interestingly, our hyperlapse videos are comparably favored by the participants against the results of 360 Hyperlapse [Lai et al. 2017], even though our system is not designed for hyperlapse videos. On the contrary, 360 Hyperlapse utilizes focus of expansion and semantic-aware frame selection for high-quality hyperlapse videos. As 360 Hyperlapse produces results with smooth camera motions as ours, most of the comments from the participants were about the contents in the videos rather than the camera motion, and the participants generally preferred videos with more contents.

5.1.2 Comparison with interactive methods. To evaluate our interactive system, we compare our system with three interactive methods: Baseline, PIP [Lin et al. 2017b], and Shot Orientation [Pavel et al. 2017]. Baseline is a conventional interactive method used in many 360° video applications such as Youtube, which allows users to change the viewing direction by dragging a mouse, but does not

[2]https://aliensunmin.github.io/project/360video/
[3]https://www.cs.utexas.edu/~ycsu/projects/watchable360/
[4]http://vllab.ucmerced.edu/wlai24/360hyperlapse/

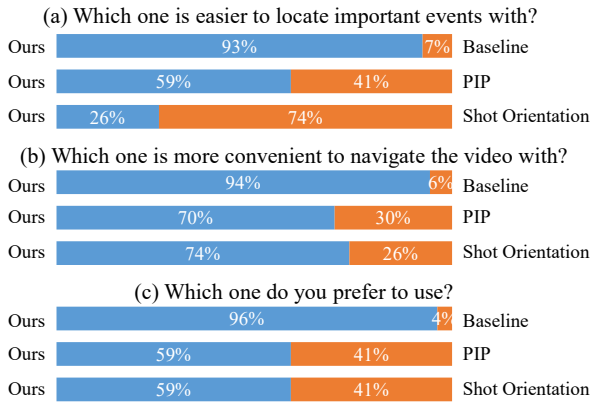


Fig. 17. User study result on comparison with other interactive systems.

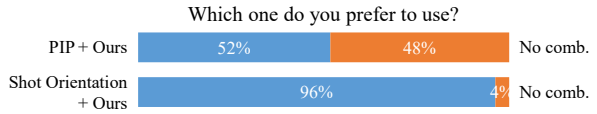


Fig. 18. User study result on combinations of other interactive systems and ours.

provide any other assistance. PIP shows picture-in-picture previews of off-screen regions of interest. The system also provides users a visual cue of the distance and direction of a region of interest using the position and tilting of its preview. A user can change the viewing direction directly to an important event by clicking a preview. Shot Orientation provides a simple user assistance such that a user can simply press a button to change the viewing direction to a region of interest that is either pre-defined or detected. For evaluation, we implemented PIP and Shot Orientation, as their code is not available.

We prepared six 360° videos and recruited 18 participants. We grouped the participants into two groups of nine people. Both groups were asked to use Baseline, PIP and ours, and Baseline, Shot Orientation and ours, respectively in random order, for three randomly chosen videos. Then, the participants were asked three questions: (a) which method is easier to locate important events with? (b) which one is more convenient to navigate the video with? and (c) which one do you prefer to use? For each question, the participants was asked to rank the three systems in an order. We then counted how many times one method was preferred over another.

Fig. 17 summarizes the user study result. Against Baseline, our system was preferred by the participants by large margins in all three questions, proving that our system clearly improves the user experience over Baseline. Against PIP [Lin et al. 2017b], our system was preferred in all three questions too, even though PIP directly shows previews of important events. A possible reason is that the size of previews can be sometimes too small to recognize for complex scenes. On the other hand, Shot Orientation [Pavel et al. 2017] was preferred by the participants in question (a), as users can quickly switch the viewing direction between different salient events by pressing a button. However, in the other questions regarding the convenience of navigation and overall preference, our system was still preferred by the participants.

We investigate the potential synergy of combinations of previous interactive systems and ours. We implemented two combinations:

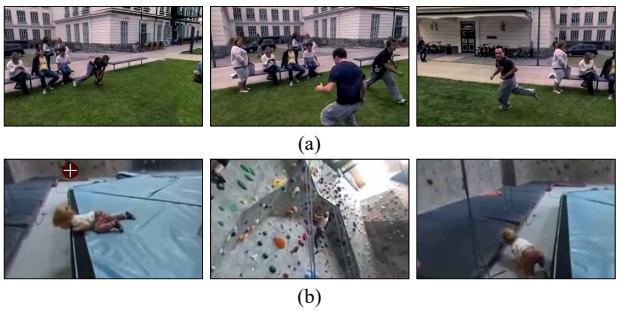


Fig. 19. Failure cases. (a) Our system may fail to track one object when there are multiple salient objects close to each other. (b) It may also change the viewing direction to a salient object against the intention of a user. Original video of (a) from Sports-360 (_evOm6vbgtU ©Lorenz Pritz).

PIP + ours, and Shot Orientation + ours. We asked the first group of the participants to use PIP + ours, and the second group to use Shot Orientation + ours. For each combination, the participants were asked whether they preferred the combination over either a previous interactive system or ours. Fig. 18 shows the user study result. In the case of Shot Orientation, the participants clearly preferred the combined system, proving that Shot Orientation and ours have a clear synergy. On the other hand, in the case of PIP, the preference was unclear. Some participants commented that the positions of previews changed too quickly in the combined system. This is because the distances and directions of regions of interest from the current viewing direction are constantly changed by our method in the combined system.

5.2 Limitations

Our system has a few limitations. As it depends on saliency and optical flow, it may fail when either one of them fails, e.g., salience estimation may fail to detect interesting events, and optical flow may fail to estimate motions of fast-moving small objects, and get confused when multiple objects occlude each other. In Fig. 19(a), multiple salient people run around and cross each other. As a result, our system frequently changes the viewing direction from one person to another. To resolve this, we may employ more sophisticated saliency estimation and object tracking methods.

Our system shows a certain direction for a certain amount of time even if there are no salient objects at the direction, then changes the viewing direction to a salient object. This behavior may sometimes not match the intention of a user. For example, in Fig. 19(b), our system originally shows a kid, and a user changes the viewing direction to a climber on the wall who is less salient. The system shows him for a while, and automatically changes back the viewing direction to the kid, which can be against the user's intention. To resolve this issue, we may interactively change the user interaction parameter α or introduce an additional lock-on mode. Video examples of failure cases can be found in the supplementary material.

Our system cannot handle interesting events happening on the top or bottom parts of a video, as we exclude those parts when computing an optimal path. We are also planning to adopt cubemaps to resolve this as done in [Kopf 2016]. Our system sometimes shows the torso instead of a person's face. This issue may be resolved by using an improved saliency estimator or a face detector. Incorporating

zooming as done in [Su and Grauman 2017b] is another interesting future work.

6 CONCLUSION

Although 360° videos are getting popular and widely available, viewing a 360° video on a 2D display is still not as comfortable as conventional 2D videos as viewers need to constantly adjust the viewing direction. In this paper, we proposed a practical and interactive system for viewing 360° video on a 2D display. Our system estimates optical flow and saliency in the pre-processing step. Based on them, the system automatically finds a camera path for the best viewing experience. While watching a video, a user can change the viewing direction, then the system instantly updates the camera path reflecting the intention of the user. The effectiveness of our system has been verified through extensive user studies.

ACKNOWLEDGMENTS

This research was supported by Basic Science Research Program and Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT (NRF-2018R1A5A1060031, NRF-2017M3C4A7066316). It was also supported by the DGIST Start-up Fund Program (2019010063).

REFERENCES

Marc Assens, Xavier Giro-i Nieto, Kevin McGuinness, and Noel E O'Connor. 2017. SaltiNet: Scan-Path Prediction on 360 Degree Images Using Saliency Volumes. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*. 2331–2338.

Jiamin Bai, Aseem Agarwala, Maneesh Agrawala, and Ravi Ramamoorthi. 2014. User-Assisted Video Stabilization. *Comput. Graph. Forum* 33, 4 (2014), 61–70.

Hsien-Tzu Cheng, Chun-Hung Chao, Jin-Dong Dong, Hao-Kai Wen, Tyng-Luh Liu, and Min Sun. 2018. Cube Padding for Weakly-Supervised Saliency Prediction in 360° Videos. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1420–1429.

Thomas Deselaers, Philippe Dreuw, and Hermann Ney. 2008. Pan, zoom, scan - Time-coherent, trained automatic video cropping. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*. 1–8.

Michael L Gleicher and Feng Liu. 2007. Re-cinematography: Improving the Camera Dynamics of Casual Video. In *Proceedings of the 15th ACM International Conference on Multimedia (MM '07)*. 27–36.

Michael L Gleicher and Feng Liu. 2008. Re-cinematography: Improving the Camerawork of Casual Video. *ACM Trans. Multimedia Comput. Commun. Appl.* 5, 1, Article 2 (2008), 28 pages.

Amit Goldstein and Raanan Fattal. 2012. Video Stabilization Using Epipolar Geometry. *ACM Trans. Graph.* 31, 5, Article 126 (2012), 10 pages.

Matthias Grundmann, Vivek Kwatra, and Irfan Essa. 2011. Auto-directed video stabilization with robust L1 optimal camera paths. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. 225–232.

Hou-Ning Hu, Yen-Chen Lin, Ming-Yu Liu, Hsien-Tzu Cheng, Yung-Ju Chang, and Min Sun. 2017. Deep 360 pilot: Learning a deep agent for piloting through 360 sports video. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1396–1405.

Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. 2017. FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1647–1655.

Junho Jeon, Jinwoong Jung, and Seungyong Lee. 2018. Deep Upright Adjustment of 360 Panoramas using Multiple Roll Estimations. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*.

Wei Jiang, Zhenyu Wu, John Wus, and Heather Yu. 2014. One-Pass Video Stabilization on Mobile Devices. In *Proceedings of the 22Nd ACM International Conference on Multimedia (MM '14)*. 817–820.

Jinwoong Jung, Beomseok Kim, Joon-Young Lee, Byungmoon Kim, and Seungyong Lee. 2017. Robust Upright Adjustment of 360 Spherical Panoramas. *Vis. Comput.* 33, 6-8 (2017), 737–747.

Yeong Won Kim, Chang-Ryeol Lee, Dae-Yong Cho, Yong Hoon Kwon, Hyeok-Jae Choi, and Kuk-Jin Yoon. 2017. Automatic Content-Aware Projection for 360deg Videos. In *2017 IEEE International Conference on Computer Vision (ICCV)*. 4753–4761.

Johannes Kopf. 2016. 360° Video Stabilization. *ACM Trans. Graph.* 35, 6, Article 195 (2016), 9 pages.

Wei-Sheng Lai, Yujia Huang, Neel Joshi, Christopher Buehler, Ming-Hsuan Yang, and Sing Bing Kang. 2017. Semantic-driven generation of hyperlapse from 360 video. *IEEE Transactions on Visualization and Computer Graphics, PP (99)* (2017), 1–1.

Yen-Chen Lin, Yung-Ju Chang, Hou-Ning Hu, Hsien-Tzu Cheng, Chi-Wen Huang, and Min Sun. 2017a. Tell Me Where to Look: Investigating Ways for Assisting Focus in 360° Video. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. 2535–2545.

Yung-Ta Lin, Yi-Chi Liao, Shan-Yuan Teng, Yi-Ju Chung, Liwei Chan, and Bing-Yu Chen. 2017b. Outside-In: Visualizing Out-of-Sight Regions-of-Interest in a 360° Video Using Spatial Picture-in-Picture Previews. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17)*. 255–265.

Ce Liu. 2009. *Beyond pixels: exploring new representations and applications for motion analysis*. Ph.D. Dissertation. Massachusetts Institute of Technology.

Feng Liu and Michael Gleicher. 2006. Video Retargeting: Automating Pan and Scan. In *Proceedings of the 14th ACM International Conference on Multimedia (MM '06)*. 241–250.

Feng Liu, Michael Gleicher, Hailin Jin, and Aseem Agarwala. 2009. Content-preserving Warps for 3D Video Stabilization. *ACM Trans. Graph.* 28, 3, Article 44 (2009), 9 pages.

Feng Liu, Michael Gleicher, Jue Wang, Hailin Jin, and Aseem Agarwala. 2011. Subspace Video Stabilization. *ACM Trans. Graph.* 30, 1, Article 4 (2011), 10 pages.

Shuaicheng Liu, Ping Tan, Lu Yuan, Jian Sun, and Bing Zeng. 2016. In *European Conference on Computer Vision*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). 800–815.

Shuaicheng Liu, Lu Yuan, Ping Tan, and Jian Sun. 2013. Bundled Camera Paths for Video Stabilization. *ACM Trans. Graph.* 32, 4, Article 78 (2013), 10 pages.

Yasuyuki Matsushita, Eyal Ofek, Weina Ge, Xiaoou Tang, and Heung-Yeung Shum. 2006. Full-frame video stabilization with motion inpainting. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28, 7 (2006), 1150–1163.

Amy Pavel, Björn Hartmann, and Maneesh Agrawala. 2017. Shot Orientation Controls for Interactive Cinematography with 360 Video. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17)*. 289–297.

Michael Rubinstein, Ariel Shamir, and Shai Avidan. 2008. Improved Seam Carving for Video Retargeting. *ACM Trans. Graph.* 27, 3, Article 16 (2008), 9 pages.

Michael Rubinstein, Ariel Shamir, and Shai Avidan. 2009. Multi-operator Media Retargeting. *ACM Trans. Graph.* 28, 3, Article 23 (2009), 11 pages.

Yu-Chuan Su and Kristen Grauman. 2017a. Learning Spherical Convolution for Fast Features from 360°Imagery. In *Advances in Neural Information Processing Systems 30*. 529–539.

Yu-Chuan Su and Kristen Grauman. 2017b. Making 360° Video Watchable in 2D: Learning Videography for Click Free Viewing. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1368–1376.

Yu-Chuan Su and Kristen Grauman. 2018. Learning Compressible 360° Video Isomers. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7824–7833.

Yu-Chuan Su, Dinesh Jayaraman, and Kristen Grauman. 2016. Pano2Vid: Automatic Cinematography for Watching 360° Videos. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*. 154–171.

Yu-Shuen Wang, Jen-Hung Hsiao, Olga Sorkine, and Tong-Yee Lee. 2011. Scalable and Coherent Video Resizing with Per-frame Optimization. *ACM Trans. Graph.* 30, 4, Article 88 (2011), 8 pages.

Yu-Shuen Wang, Hui-Chih Lin, Olga Sorkine, and Tong-Yee Lee. 2010. Motion-based Video Retargeting with Optimized Crop-and-warp. *ACM Trans. Graph.* 29, 4, Article 90 (2010), 9 pages.

Lior Wolf, Moshe Guttmann, and Daniel Cohen-Or. 2007. Non-homogeneous Content-driven Video-retargeting. In *2007 IEEE 11th International Conference on Computer Vision*. 1–6.

Stephen Wright and Jorge Nocedal. 2006. *Numerical Optimization* (2 ed.).

Feng Zhou, Sing Bing Kang, and Michael F Cohen. 2014. Time-mapping using space-time saliency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3358–3365.