Package: ovideo (via r-universe)

August 12, 2024

Title Volleyball Video Utilities

Version 1.0.0

Description Playlists and other video support functions from volleyball match files.

URL https://github.com/openvolley/ovideo

BugReports https://github.com/openvolley/ovideo/issues

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

Depends datavolley (> 0.12.999)

Imports assertthat, av, base64enc, digest, dplyr, DT, fs, ggplot2, htmltools, httr, knitr, jpeg, jsonlite, R.utils, rappdirs, rmarkdown, shiny, sys

Suggests editry, future, future.apply, ovdata, remotes, testthat

Remotes openvolley/datavolley, openvolley/ovdata, scienceuntangled/editry

RoxygenNote 7.3.1

Repository https://openvolley.r-universe.dev

RemoteUrl https://github.com/openvolley/ovideo

RemoteRef HEAD

RemoteSha a5e3d9d0863b5084fe578339fbf6d88b3bc88471

Contents

browseFile 2
dv_meta_video 3
ovideo 4
ov_3dpos_multicamera 4

ov_cmat_apply 6
ov_cmat_estimate 7
ov_editry_clips 8
ov_example_video 11
ov_ffmpeg_exe 11
ov_find_video_file 12
ov_get_court_ref 12
ov_get_video_data 13
ov_get_video_meta 14
ov_images_to_video 15
ov_install_ffmpeg 16
ov_merge_video_timing_df 17
ov_overlay_data 17
ov_overlay_image 18
ov_playlist_as_onclick 19
ov_playlist_to_html 20
ov_playlist_to_video 22
ov_playlist_to_vlc 24
ov_set_video_data 25
ov_set_video_meta 26
ov_shiny_court_ref 27
ov_transform_points 28
ov_video_control 30
ov_video_extract_clip 30
ov_video_frame 31
ov_video_frames 32
ov_video_js 33
ov_video_player 34
ov_video_playlist 35
ov_video_playlist_pid 37
ov_video_timing 38

Index **40**

`browseFile` *Browse a system file in the default browser*

Description

RStudio overrides the default behaviour of browseURL on some platforms, meaning that local files are not opened as `file:///...` URLs but as `http://localhost....` This can break some local HTML files that are expecting to be served as `file:///` URLs.

Usage

```
browseFile(url, browser = getOption("browser"), encodeIfNeeded = FALSE)
```

Arguments

url	string: as for `utils::browseURL()`
browser	string: as for `utils::browseURL()`
encodeIfNeeded	logical: as for `utils::browseURL()`

Examples

```
myfile <- tempfile(fileext = ".html")
cat("<h1>Hello!</h1>", file = myfile)

## in RStudio on Linux, this will be opened as a http://localhost URL
if (interactive()) browseURL(myfile)

## but this shouldn't
browseFile(myfile)
```

dv_meta_video *Get or set the video metadata in a datavolley object*

Description

Get or set the video metadata in a datavolley object

Usage

```
dv_meta_video(x)

dv_meta_video(x) <- value
```

Arguments

x	datavolley: a datavolley object as returned by `datavolley::dv_read()`
value	string or data.frame: a string containing the path to the video file, or a data.frame with columns "camera" and "file"

Value

For `dv_meta_video`, the existing video metadata. For `dv_meta_video<-`, the video metadata value in `x` is changed

Examples

```
x <- dv_read(dv_example_file())
dv_meta_video(x) ## empty dataframe
dv_meta_video(x) <- "/path/to/my/videofile"
dv_meta_video(x)
```

`ovideo` **ovideo**

Description

Playlists and other video support functions from volleyball match files.

Author(s)

Maintainer: Ben Raymond `<ben@untan.gl>`

Authors:

- Adrien Ickowicz

Other contributors:

- openvolley.org [originator]

See Also

Useful links:

- `https://github.com/openvolley/ovideo`
- Report bugs at `https://github.com/openvolley/ovideo/issues`

`ov_3dpos_multicamera` *3D position estimate from multiple 2D views*

Description

3D position estimate from multiple 2D views

Usage

```
ov_3dpos_multicamera(uv, C, method = "dlt", zinit = 2)
```

Arguments

`uv`	matrix or data.frame: u, v positions in 2D images, one row per image (u and v are the image x- and y-coordinates, normalized to the range 0-1)
`C`	list: a list of the same length as the number of rows in uv. The ith entry of `C` is the camera matrix (as returned by `ov_cmat_estimate()`) associated with the image coordinates in row i of uv. NOTE that the camera matrices in C must all return positions with the same orientation (i.e. all on the same court coordinates, oriented the same way)

`method`	string: either "dlt" (direct linear transform) or "nls" (nonlinear least-squares). The "nls" method finds the real-world x and y coordinates for each point in `uv`, assuming a certain value of z. It then chooses z to minimize the difference between those real-world x, y positions
`zinit`	numeric: initial estimate of height (only for `method = "nls"`)

Value

A named list with components `xyz` (the estimated 3D position) and `err` (a measure of uncertainty in that position estimate - currently only for method "nls")

References

For general background see e.g. Ballard DH, Brown CM (1982) Computer Vision. Prentice-Hall, New Jersey

Examples

```
 ## two camera matrices
 refpts1 <- dplyr::tribble(~image_x, ~image_y, ~court_x, ~court_y, ~z,
                            0.0533,   0.0326,      3.5,      6.5,  0,
                             0.974,   0.0572,      0.5,      6.5,  0,
                             0.683,    0.566,      0.5,      0.5,  0,
                             0.283,    0.560,      3.5,      0.5,  0,
                             0.214,    0.401,      3.5,      3.5,  0,
                             0.776,    0.412,      0.5,      3.5,  0,
                             0.780,    0.680,      0.5,      3.5,  2.43,
                             0.206,    0.670,      3.5,      3.5,  2.43)

 C1 <- ov_cmat_estimate(x = refpts1[, c("image_x", "image_y")],
                        X = refpts1[, c("court_x", "court_y", "z")])

 refpts2 <- dplyr::tribble(~image_x, ~image_y, ~court_x, ~court_y, ~z,
                             0.045,   0.0978,      0.5,      0.5,  0,
                             0.963,   0.0920,      3.5,      0.5,  0,
                             0.753,    0.617,      3.5,      6.5,  0,
                             0.352,    0.609,      0.5,      6.5,  0,
                             0.255,    0.450,      0.5,      3.5,  0,
                             0.817,    0.456,      3.5,      3.5,  0,
                             0.821,    0.731,      3.5,      3.5,  2.43,
                             0.246,    0.720,      0.5,      3.5,  2.43)
 C2 <- ov_cmat_estimate(x = refpts2[, c("image_x", "image_y")],
                        X = refpts2[, c("court_x", "court_y", "z")])

# uv1 <- ov_cmat_apply(C1, matrix(xyz, ncol = 3))c(0.369, 0.775) ## object position in image 1
# uv2 <- c(0.732, 0.688) ## object position in image 2

xyz <- matrix(c(3.4, 1.4, 2.90), ncol = 3)
uv1 <- ov_cmat_apply(C1, xyz) ## object position in image 1
uv2 <- ov_cmat_apply(C2, xyz) ## object position in image 2

## if our measurements are perfect (no noise), we can reconstruct xyz exactly:
```

```
ov_3dpos_multicamera(rbind(uv1, uv2), list(C1, C2), method = "dlt")
ov_3dpos_multicamera(rbind(uv1, uv2), list(C1, C2), method = "nls")

## with noise
uv1 <- uv1 + rnorm(2, sd = 0.02)
uv2 <- uv2 + rnorm(2, sd = 0.02)
ov_3dpos_multicamera(rbind(uv1, uv2), list(C1, C2), method = "dlt")
ov_3dpos_multicamera(rbind(uv1, uv2), list(C1, C2), method = "nls")
```

---

ov_cmat_apply *Apply the camera matrix to 3D coordinates*

Description

The camera matrix characterizes the mapping of a camera from 3D real-world coordinates to 2D coordinates in an image.

Usage

```
ov_cmat_apply(C, X)
```

Arguments

`C`	: camera matrix as returned by `ov_cmat_estimate()`, or the coefficients from that object
`X`	matrix or data.frame: Nx3 matrix of 3D real-world coordinates

Value

An Nx2 matrix of image coordinates

References

https://en.wikipedia.org/wiki/Camera_matrix. For general background see e.g. Ballard DH, Brown CM (1982) Computer Vision. Prentice-Hall, New Jersey

See Also

`ov_cmat_estimate()`

Examples

```
## define real-world and corresponding image coordinates
xX <- dplyr::tribble(~image_x, ~image_y, ~court_x, ~court_y,   ~z,
                        0.054,    0.023,      0.5,      0.5,    0, ## near left baseline
                        0.951,    0.025,      3.5,      0.5,    0, ## near right baseline
                        0.752,    0.519,      3.5,      6.5,    0, ## far right baseline
                        0.288,    0.519,      0.5,      6.5,    0, ## far left baseline
                        0.199,    0.644,      0.5,      3.5, 2.43, ## left net top
                        0.208,    0.349,      0.5,      3.5, 0.00, ## left net floor
                        0.825,    0.644,      3.5,      3.5, 2.43, ## right net top
                        0.821,    0.349,      3.5,      3.5, 0.00) ## right net floor

C <- ov_cmat_estimate(X = xX[, 3:5], x = xX[, 1:2])

## fitted image coordinates using C
ov_cmat_apply(C, X = xX[, 3:5])

## compare to actual image positions
xX[, 1:2]
```

`ov_cmat_estimate` *Estimate the camera matrix*

Description

The camera matrix characterizes the mapping of a camera from 3D real-world coordinates to 2D coordinates in an image.

Usage

```
ov_cmat_estimate(X, x)
```

Arguments

X	matrix or data.frame: Nx3 matrix of 3D real-world coordinates
x	matrix or data.frame: Nx2 matrix of image coordinates

Value

A list with components `coef` (fitted transformation coefficients) and `rmse` (root mean squared error of the fitted transformation)

References

https://en.wikipedia.org/wiki/Camera_matrix. For general background see e.g. Ballard DH, Brown CM (1982) Computer Vision. Prentice-Hall, New Jersey

See Also

`ov_cmat_apply()`

Examples

```
## define real-world and corresponding image coordinates
xX <- dplyr::tribble(~image_x, ~image_y, ~court_x, ~court_y,   ~z,
                        0.054,    0.023,      0.5,      0.5,    0, ## near left baseline
                        0.951,    0.025,      3.5,      0.5,    0, ## near right baseline
                        0.752,    0.519,      3.5,      6.5,    0, ## far right baseline
                        0.288,    0.519,      0.5,      6.5,    0, ## far left baseline
                        0.199,    0.644,      0.5,      3.5, 2.43, ## left net top
                        0.208,    0.349,      0.5,      3.5, 0.00, ## left net floor
                        0.825,    0.644,      3.5,      3.5, 2.43, ## right net top
                        0.821,    0.349,      3.5,      3.5, 0.00) ## right net floor

C <- ov_cmat_estimate(X = xX[, 3:5], x = xX[, 1:2])

## fitted image coordinates using C
ov_cmat_apply(C, X = xX[, 3:5])

## compare to actual image positions
xX[, 1:2]
```

---

ov_editry_clips *Convert playlist to editry clips*

Description

Note that in order to use ov_editry_clips, the editry package must be installed. Install it with: remotes::install_github('scienceuntangled/editry') or install.packages('editry', repos = c('https://openvolley.r-universe.dev', 'https://cloud.r-project.org')). The editry package also requires editly (the underlying node JS package: see editry::er_install_editly()).

Usage

```
ov_editry_clips(
  playlist,
  title = NULL,
  title2 = NULL,
  label_col,
  pause = TRUE,
  seamless = FALSE,
  title_args = list(),
  title2_args = list(),
  pause_args = list(),
  label_args = list()
)
```

Arguments

`playlist`	data.frame: a playlist as returned by `ov_video_playlist`. Note that only local video sources are supported
`title`	string: the title text (first slide). Use `NULL` to skip this slide
`title2`	string: the second title text (on the second slide). Use `NULL` to skip this slide
`label_col`	string: the name of the column in `playlist` to use for a label on each clip (created with `editry::er_layer_news_title()`). Common label choices are player names, scores, or descriptions of the play being shown in the clip
`pause`	logical: if `TRUE`, insert an `editry::er_clip_pause()` clip at the end of the sequence
`seamless`	logical: if `TRUE`, combine overlapping/adjacent clips. Note that if a `label_col` has been specified, the label from the first clip will be used for the whole of the combined clip (so it may no longer make sense, for example if the label is the player name)
`title_args`	list: arguments to pass to `editry::er_clip_title_background()` when creating the title slide
`title2_args`	list: arguments to pass to `editry::er_clip_title2()` when creating the title2 slide
`pause_args`	list: arguments to pass to `editry::er_clip_pause()` when creating the final slide
`label_args`	list: arguments to pass to `editry::er_layer_news_title()`, used if `label_col` is provided

Value

A list of `editry::er_clip()` objects, suitable to pass to `editry::er_spec()`

See Also

`editry::er_layer_news_title()`, `editry::er_layer()`, `editry::er_spec()`

Examples

```
## Not run:
  ## Example 1

  ## Step 1: create our playlist

  ## use data from the ovdata package
  library(ovdata) ## install via remotes::install_github("openvolley/ovdata") if needed
  x <- ovdata_example("190301_kats_beds-clip", as = "parsed")

  ## make sure its video element points to our local copy of the corresponding video clip
  dv_meta_video(x) <- ovdata_example_video("190301_kats_beds")

  ## extract the plays
  px <- datavolley::plays(x)
  ## use just the attack rows
  px <- px[which(px$skill == "Attack"), ]

  ## make a new column with player name and attack type
  px$label <- paste(px$player_name, px$attack_code, "attack")

  ## make the playlist with the new label column included
  tm <- ov_video_timing(Attack = c(-3, 2)) ## tighter than normal timing
  ply <- ov_video_playlist(px, x$meta, extra_cols = "label", timing = tm)

  ## Step 2: convert to editly clip objects and compile to mp4

  library(editry)
  ## create the clips, one for each row of the playlist
  clips <- ov_editry_clips(ply, title = "GKS Katowice\nvs\nMKS Bedzin",
                              title2 = "Attacks", label_col = "label")


  ## compile to video
  outfile <- tempfile(fileext = ".mp4")
  my_spec <- er_spec(out_path = outfile, clips = clips)
  er_exec_wait(spec = my_spec, fast = TRUE)

  ## and view the output
  if (interactive()) browseURL(outfile)

  ## ---

  ## Example 2
  ## without a playlist, make a simple clip from a known segment of video

  library(editry)
  library(ovdata) ## install via remotes::install_github("openvolley/ovdata") if needed
  my_video <- ovdata_example_video("190301_kats_beds") ## path to your video file
 my_logo <- "https://github.com/openvolley/community/raw/master/docs/talks/common/ovlogo-blur.png"

 clips <- list(er_clip_video(path = my_video, cut_from = 1, cut_to = 8), ## video segment
              ## add an outro banner with logo
              er_clip(duration = 1.5, layers = list(er_layer_fill_color(),
                                                      er_layer_image(path = my_logo))),
              ##  and blank finishing screen
              er_clip_pause(duration = 0.25))

  outfile <- tempfile(fileext = ".mp4")
  my_spec <- er_spec(clips = clips, out_path = outfile, allow_remote_requests = TRUE)

  er_exec_wait(spec = my_spec, fast = TRUE)
  if (interactive()) browseURL(outfile)

## End(Not run)
```

`ov_example_video` *Example video clips provided as part of the ovideo package*

Description

Example video clips provided as part of the ovideo package

Usage

```
ov_example_video(choice = 1)
```

Arguments

`choice`	integer: which video file to return? • 1 - a clip from a match between GKS Katowice and MKS Bedzin during the 2018/19 Polish Plus Liga

Value

Path to the video file

`ov_ffmpeg_exe` *ffmpeg executable functions*

Description

Helper functions to find the ffmpeg executable. If ffmpeg is not installed on the system, it can be installed (for some platforms) with `ov_install_ffmpeg()`.

Usage

```
ov_ffmpeg_exe()

ov_ffmpeg_ok(do_error = FALSE)
```

Arguments

`do_error`	logical: if `TRUE`, throw an error if the ffmpeg executable cannot be found

Value

For `ov_ffmpeg_exe`, the path to the executable, or `NULL` if not found. For `ov_ffmpeg_ok`, a logical indicating whether the executable could be found or not

See Also

`ov_install_ffmpeg()`

Examples

```
ov_ffmpeg_ok()
```

`ov_find_video_file` *Try and locate a video file, when the path embedded in the dvw file is for another computer*

Description

Try and locate a video file, when the path embedded in the dvw file is for another computer

Usage

```
ov_find_video_file(dvw_filename, video_filename = NULL)
```

Arguments

`dvw_filename`	string: the full path to the DataVolley file
`video_filename`	character: one or more video file paths. If `NULL`, the video file name embedded in the DataVolley file will be used

Value

A character vector, with one entry per `video_filename`. Video files that could not be found will be `NA` here.

`ov_get_court_ref` *Define the reference points on a court image*

Description

This function is used to define the reference points on a court image, to be used with `ov_transform_points()`. The court coordinate system is that used in `datavolley::dv_court()`, `datavolley::ggcourt()`, and related functions. Try `plot(c(0, 4), c(0, 7), type = "n", asp = 1); datavolley::dv_court()` or `ggplot2::ggplot() + datavolley::ggcourt() + ggplot2::theme_bw()` for a visual depiction.

Usage

```
ov_get_court_ref(image_file, video_file, t = 60, type = "corners")
```

Arguments

`image_file`	string: path to an image file (jpg) containing the court image (not required if `video_file` is supplied)
`video_file`	string: path to a video file from which to extract the court image (not required if `image_file` is supplied)
`t`	numeric: the time of the video frame to use as the court image (not required if `image_file` is supplied)
`type`	string: currently only "corners"

Value

A data.frame containing the reference information

See Also

`ov_transform_points()`, `datavolley::dv_court()`, `datavolley::ggcourt()`

Examples

```
if (interactive()) {
 crt <- ov_get_court_ref(image_file = system.file("extdata/2019_03_01-KATS-BEDS-court.jpg",
                         package = "ovideo"))

}
```

`ov_get_video_data` *Retrieve a data object stored in a video file metadata tag*

Description

Retrieve a data object stored in a video file metadata tag

Usage

```
ov_get_video_data(video_file, tag = "ov_court_info", b64 = TRUE)
```

Arguments

`video_file`	string: path to the video file
`tag`	string: the tag name to use
`b64`	logical: was `obj` serialized and base64-encoded before storing?

Value

The stored information, or `NULL` if there was none

See Also

`ov_set_video_data()`

Examples

```
## Not run:
  if (interactive()) {
    ## mark the geometry of the court in the video
    ref <- ov_shiny_court_ref(video_file = ov_example_video(), t = 5)

    ## store it
    newfile <- ov_set_video_data(ov_example_video(), obj = ref)

    ## retrieve it
    ov_get_video_data(newfile)
  }

## End(Not run)
```

`ov_get_video_meta` *Retrieve metadata tags from a video file*

Description

Requires that ffmpeg is available on your system path.

Usage

```
ov_get_video_meta(video_file, debug = FALSE)
```

Arguments

`video_file`	string: path to the video file
`debug`	logical: if `TRUE`, echo the ffmpeg output to the console

Value

A named list of metadata values

See Also

`ov_set_video_meta()`

Examples

```
## Not run:
  newfile <- ov_set_video_meta(ov_example_video(), comment = "A comment")
  ov_get_video_meta(newfile)

## End(Not run)
```

`ov_images_to_video` *Encode a set of images into a video*

Description

Requires that ffmpeg is available on your system path. Input files can either be specified as a list of image files, or alternatively as a directory name and image file mask. For the latter, the images must be numbered in sequential order.

Usage

```
ov_images_to_video(
  input_dir,
  image_file_mask = "image_%06d.jpg",
  image_files,
  outfile,
  fps = 30,
  extra = NULL,
  debug = FALSE
)
```

Arguments

`input_dir`	string: path to the input directory
`image_file_mask`	string: the mask that specifies the image files, e.g. "image_%06d.jpg" for images named "image_000001.jpg", "image_000002.jpg" etc
`image_files`	character: vector of input image files, in order that they should appear in the video. Used if `input_dir` is missing
`outfile`	string: the output file. If missing, a temporary file (with extension .mp4) will be used
`fps`	numeric: frames per second
`extra`	: additional parameters passed to ffmpeg, in the form c("param", "value", "param2", "value2"). For example, `c("-vb", "4096k")` could be used to control the output video bitrate
`debug`	logical: if `TRUE`, echo the ffmpeg output to the console

Value

The path to the video file

See Also

`av::av_encode_video()` as an alternative

`ov_install_ffmpeg` *Install ffmpeg*

Description

This is a helper function to install ffmpeg. Currently it only works on Windows and Linux platforms. The ffmpeg bundle will be downloaded from `https://github.com/BtbN/FFmpeg-Builds/releases/latest` (Windows) or `https://johnvansickle.com/ffmpeg/` (Linux) and saved to your user appdata directory.

Usage

```
ov_install_ffmpeg(force = FALSE, bits, check_hash = TRUE)
```

Arguments

`force`	logical: force reinstallation if ffmpeg already exists
`bits`	integer: 32 or 64, for 32- or 64-bit install. If missing or `NULL`, will be guessed based on `.Machine$sizeof.pointer`. Note that only 64-bit is supported on Windows
`check_hash`	logical: don't check the hash of the downloaded file. Ignored on windows

Value

the path to the installed executable

References

`https://github.com/BtbN/FFmpeg-Builds/releases/latest` `https://johnvansickle.com/ffmpeg/`

Examples

```
## Not run:
  ov_install_ffmpeg()

## End(Not run)
```

`ov_merge_video_timing_df` *Merge two video timing dataframes*

Description

Merge two video timing dataframes

Usage

```
ov_merge_video_timing_df(x, default = ov_video_timing_df())
```

Arguments

`x`	data.frame: video timings to use
`default`	data.frame: default timings to use, for anything not provided in `x`

Value

A data.frame

See Also

`ov_video_timing_df()`

Examples

```
my_timings <- data.frame(skill = "Attack", phase = "Reception", start_offset = 0)
ov_merge_video_timing_df(my_timings)
```

`ov_overlay_data` *Generate data suitable for creating a court overlay plot*

Description

Generate data suitable for creating a court overlay plot

Usage

```
ov_overlay_data(
  zones = TRUE,
  serve_zones = TRUE,
  labels = FALSE,
  space = "court",
  court_ref,
  crop = TRUE
)
```

Arguments

`zones`	logical: if `TRUE`, show zone lines
`serve_zones`	logical: if `TRUE`, show the serve zones behind the baselines
`labels`	logical: if `TRUE`, label the zones
`space`	string: if "court", the data will be in court coordinates. If "image", the data will be transformed to image coordinates via `ov_transform_points`
`court_ref`	data.frame: as returned by `ov_get_court_ref`. Only required if `space` is "image"
`crop`	logical: if `space` is "image", and `crop` is TRUE, the data will be cropped to the c(0, 1, 0, 1) bounding box (i.e. the limits of the image, in normalized coordinates)

Value

A list of data.frames

See Also

`ov_overlay_image`

---

ov_overlay_image — *Generate a court overlay image showing court boundary, 3m, zone, and other lines*

Description

Generate a court overlay image showing court boundary, 3m, zone, and other lines

Usage

```
ov_overlay_image(court_ref, height, width, filename, ...)
```

Arguments

`court_ref`	data.frame: as returned by `ov_get_court_ref`
`height`	integer: height of image to produce in pixels
`width`	integer: width of image to produce in pixels
`filename`	string: image filename (png). If missing, a file will be created in the temporary directory
`...`	: arguments passed to `ov_overlay_data`

Value

The path to the generated file.

ov_playlist_as_onclick

Convert playlist to 'onclick' string

Description

Convert playlist to 'onclick' string

Usage

```
ov_playlist_as_onclick(
  playlist,
  video_id,
  normalize_paths = TRUE,
  dvjs_fun = "dvjs_set_playlist_and_play",
  seamless = TRUE,
  loop = FALSE,
  controller_var
)
```

Arguments

`playlist`	data.frame: a playlist as returned by `ov_video_playlist`
`video_id`	string: the id of the HTML video element to attach the playlist to
`normalize_paths`	logical: if `TRUE`, apply `normalizePath` to local file paths. This will e.g. expand the tilde in paths like "~/path/to/video.mp4"
`dvjs_fun`	string: the javascript function to use
`seamless`	logical: if clips overlap, should we transition seamlessly from one to the next?
`loop`	logical: should we loop endlessly over the playlist?
`controller_var`	string: (for version 2 only) the js variable name of the controller object to assign this playlist to

Value

A string suitable for inclusion as an 'onclick' tag attribute

Examples

```
## Not run:
  library(shiny)

  ## hand-crafted playlist for this example
  playlist <- data.frame(video_src = "NisDpPFPQwU",
                         start_time = c(624, 3373, 4320),
                         duration = 8,
                       type = "youtube")
  shinyApp(
      ui = fluidPage(
          ov_video_js(youtube = TRUE),
          ov_video_player(id = "yt_player", type = "youtube",
                          style = "height: 480px; background-color: black;"),
          tags$button("Go", onclick = ov_playlist_as_onclick(playlist, "yt_player"))
      ),
      server = function(input, output) {},
  )

  ## or using v2, which supports multiple video elements in a page
  shinyApp(
      ui = fluidPage(
          ov_video_js(youtube = TRUE, version = 2),
          ## first player
          ov_video_player(id = "yt_player", type = "youtube",
                          style = "height: 480px; background-color: black;",
                          version = 2, controller_var = "my_dv"),
          tags$button("Go", onclick = ov_playlist_as_onclick(playlist, "yt_player",
                                                             controller_var = "my_dv")),
          ## second player
          ov_video_player(id = "yt_player2", type = "youtube",
                          style = "height: 480px; background-color: black;",
                          version = 2, controller_var = "my_dv2"),
          tags$button("Go", onclick = ov_playlist_as_onclick(playlist, "yt_player2",
                                                             controller_var = "my_dv2"))
      ),
      server = function(input, output) {},
  )


## End(Not run)
```

`ov_playlist_to_html` *Convert playlist to standalone HTML file*

Description

Converts a playlist object to an HTML file that can be opened in any browser. Note that if the playlist uses local video files, the HTML file will only work on a device that has access to those files. If the playlist uses YouTube (or other external) video URLs, the HTML file will be usable on any network-connected device.

Usage

```
ov_playlist_to_html(
  playlist,
  playlist_name = "Playlist",
  outfile,
  no_paths = FALSE,
  table_cols = c(),
  loop = FALSE,
  ...
)
```

Arguments

`playlist`	data.frame: as returned by `ov_video_playlist()`. If the playlist contains one or both of the columns `subtitle` or `subtitleskill`, these will be shown as subtitle information that changes as each clip is played
`playlist_name`	string: the name to use for the playlist
`outfile`	string: the file name to write to. If not supplied, a file will be created in the temporary directory. Note that the directory of `outfile` must already exist
`no_paths`	logical: if `TRUE`, remove the paths from video files (only applicable to local files, not YouTube or other external URLs). If `no_paths` is `TRUE`, The HTML file must be saved into the same directory as the video source file(s)
`table_cols`	character: the names of columns in `playlist` to show in the plays table in the HTML file. If `table_cols` is empty, or contains no column names that are present in `playlist`, then no table will be shown
`loop`	logical: should we loop endlessly over the playlist?
`...`	: additional arguments passed to the Rmd file used to generate the HTML. Currently these are: • css : a string with additional css to apply to the file • ui_header : a tag element to replace the default page header

Value

The path to the HTML file

See Also

`ov_video_playlist()` `ov_playlist_to_vlc()`

Examples

```
## Not run:
  ## use data from the ovdata package
  library(ovdata) ## install via remotes::install_github("openvolley/ovdata") if needed
  x <- ovdata_example("190301_kats_beds-clip", as = "parsed")

  ## make sure its video element points to our local copy of the corresponding video clip
  dv_meta_video(x) <- ovdata_example_video("190301_kats_beds")

  ## extract the plays
  px <- datavolley::plays(x)
  ## it's a single rally, so we'll use all rows (just exclude NA skill rows)
  px <- px[!is.na(px$skill), ]

  ## define columns to show in the table
  extra_cols <- c("home_team", "visiting_team", "video_time", "code", "set_number",
                  "home_team_score", "visiting_team_score")

  ## make the playlist with extra columns included
  ply <- ov_video_playlist(px, x$meta, extra_cols = c(extra_cols, "player_name"))

  ## use player name as the subtitle
  ply$subtitle <- ply$player_name

  ## convert to HTML
  f <- ov_playlist_to_html(ply, table_cols = extra_cols)

  ## and finally open it!
  browseFile(f)

## End(Not run)
```

ov_playlist_to_video *Playlist to video file*

Description

Make a self-contained video file from a playlist.

Usage

```
ov_playlist_to_video(
  playlist,
  filename,
  subtitle_column = NULL,
  seamless = FALSE,
  debug = FALSE
)
```

Arguments

`playlist`	data.frame: a playlist as returned by `ov_video_playlist`. Note that only local video sources are supported
`filename`	string: file to write to. If not specified (or `NULL`), a file in the temporary directory will be created. If `filename` exists, it will be overwritten. The extension of `filename` will determine the output format
`subtitle_column`	string: if not `NULL`, a subtitle file will be produced using the contents of this column (in the playlist) as the subtitle for each clip. The subtitle file will have the same name as `filename` but with extension ".srt"

`seamless`	logical: if `TRUE`, combine overlapping/adjacent clips. Note that if a `subtitle_col` has been specified, the subtitle from the first clip will be used for the whole of the combined clip (so it may no longer make sense, for example if the subtitle is the player name)
`debug`	logical: if `TRUE`, echo the ffmpeg output to the console

Details

Requires that ffmpeg be available on the system path. Note that the processing of each clip is done inside of a `future_lapply` call (if the `future.apply` package is installed), and so you can have this part of the processing done in parallel by setting an appropriate futures plan before calling this function. This function is experimental. In particular it is unlikely to work well with all video formats, and especially if the playlist comprises clips from different videos with different resolution/encoding/etc.

Value

A list with the filenames of the created video and subtitle files.

See Also

`ov_video_playlist()`

Examples

```
## Not run:
  my_playlist <- ov_video_playlist(..., type = "local")
  video_file <- ov_create_video(my_playlist)
  browseURL(video_file[[1]])

  ## run in parallel, with the scouted codes as subtitles
  library(dplyr)
  library(future.apply)
  plan(multisession)
  ## note that the example file doesn't have a video associated with it, so
  ##  this example won't actually work in practice
  x <- read_dv(dv_example_file())
  ## fudge the video entry
  dv_meta_video(x) <- "~/my_video.mp4"
  ## make the playlist
  my_playlist <- ov_video_playlist(
    x$plays %>% dplyr::filter(skill == "Reception") %>% slice(1:10),
    meta = x$meta, extra_cols = "code")
  ## create the video and subtitles files
  video_file <- ov_create_video(my_playlist, subtitle_column = "code")

## End(Not run)
```

`ov_playlist_to_vlc` *Convert playlist to VLC m3u format*

Description

Converts a playlist object to a m3u file that can be opened with VLC. Note that this only works with local video files (not YouTube or other URLs) and the video files must be present on your local file system in order for this to work.

Usage

```
ov_playlist_to_vlc(playlist, outfile, no_paths = FALSE, seamless = TRUE)
```

Arguments

`playlist`	data.frame: as returned by `ov_video_playlist()`. If the playlist contains one or both of the columns `subtitle` or `subtitleskill`, these will be used as subtitle information associated with each clip
`outfile`	string: the file name to write to. If not supplied, a file will be created in the temporary directory. Note that the directory of `outfile` must already exist
`no_paths`	logical: if `TRUE`, remove the paths from video files. The m3u file must be saved into the same directory as the video source file(s)
`seamless`	logical: if `TRUE`, merge adjacent items into a single clip

Value

The path to the m3u file

References

https://www.videolan.org/, https://wiki.videolan.org/M3U/

See Also

`ov_video_playlist()` `ov_playlist_to_html()`

ov_set_video_data *Store a data object in a video file metadata tag*

Description

This function stores an R data object (data frame, list, etc) within a metadata tag inside a video file. This is primarily intended to store video-specific information, so that this information is carried with the video file itself. By default the `ov_court_info` metadata tag is used (intended to store the geometry of the playing court in the video, see Examples).

Usage

```
ov_set_video_data(
  video_file,
  obj,
  tag = "ov_court_info",
  b64 = TRUE,
  replace = FALSE,
  overwrite = FALSE
)
```

Arguments

`video_file`	string: path to the video file
`obj`	: data object to store, typically a list as returned by `ov_shiny_court_ref()`. `obj` will be serialized and base64-encoded before storing unless `b64 = FALSE`
`tag`	string: the tag name to use
`b64`	logical: serialize `obj` and base64-encode before storing?
`replace`	logical: if `FALSE` and the specified metadata tag is already present in the video file, don't replace it
`overwrite`	logical: if `TRUE` overwrite the `video_file`, otherwise create a new file in the temporary directory. See `ov_set_video_meta()` for details

Value

The path to the video file

See Also

`ov_get_video_data()`, `ov_set_video_meta()`

Examples

```
## Not run:
  if (interactive()) {
    ## mark the geometry of the court in the video
    ref <- ov_shiny_court_ref(video_file = ov_example_video(), t = 5)

    ## store it
    newfile <- ov_set_video_data(ov_example_video(), obj = ref)

    ## retrieve it
    ov_get_video_data(newfile)
  }

## End(Not run)
```

`ov_set_video_meta` *Set metadata tags in a video file*

Description

Requires that ffmpeg is available on your system path.

Usage

```
ov_set_video_meta(
  video_file,
  ...,
  movflags = FALSE,
  overwrite = FALSE,
  debug = FALSE
)
```

Arguments

`video_file`	string: path to the video file
`...`	: named values to set
`movflags`	logical: if `TRUE`, add "-movflags use_metadata_tags" to the command-line ffmpeg call. This allows arbitrary tag names to be used with mp4/m4v/mov video formats, but note that these may be stored in a manner that some video software cannot read. If `movflags = FALSE`, the supported video tag names (i.e. allowable names in the `...` parameters) depend on the video file type
`overwrite`	logical: if `TRUE` overwrite the `video_file`, see Details
`debug`	logical: if `TRUE`, echo the ffmpeg output to the console

Details

This function creates a new video file with the specified metadata added. This is always a file in the temporary directory. If `overwrite = TRUE`, the original file is deleted and replaced with the new file.

Note that if `movflags = FALSE`, the supported video tag names (i.e. allowable names in the `...` parameters) depend on the video file type.

Value

The path to the new video file, which if `overwrite = TRUE` will be the input file, otherwise a file in the temporary directory

See Also

`ov_get_video_meta()`

Examples

```
## Not run:
  newfile <- ov_set_video_meta(ov_example_video(), comment = "A comment")
  ov_get_video_meta(newfile)

## End(Not run)
```

`ov_shiny_court_ref` *A shiny app to define a court reference*

Description

A shiny app to define a court reference

Usage

```
ov_shiny_court_ref(
  image_file,
  video_file,
  t = 60,
  existing_ref = NULL,
  launch_browser = getOption("shiny.launch.browser", interactive()),
  ...
)
```

Arguments

`image_file`	string: path to an image file (jpg) containing the court image (not required if `video_file` is supplied)
`video_file`	string: path to a video file from which to extract the court image (not required if `image_file` is supplied)
`t`	numeric: the time of the video frame to use as the court image (not required if `image_file` is supplied)
`existing_ref`	list: (optional) the output from a previous call to `ov_shiny_court_ref()`, which can be edited
`launch_browser`	logical: if `TRUE`, launch the app in the system browser
`...`	: additional parameters (currently ignored)

Value

A list containing the reference information

Examples

```
if (interactive()) {
  ## define a court reference from scratch
  ov_shiny_court_ref(video_file = ov_example_video(), t = 5)

  ## or modify an existing one
  crt <- data.frame(image_x = c(0.05397063, 0.95402573, 0.75039756, 0.28921230),
                    image_y = c(0.02129301, 0.02294600, 0.52049712, 0.51884413),
                    court_x = c(0.5, 3.5, 3.5, 0.5),
                    court_y = c(0.5, 0.5, 6.5, 6.5))
  ref <- list(court_ref = crt, net_height = 2.43)
  ov_shiny_court_ref(video_file = ov_example_video(), t = 5, existing_ref = ref)
}
```

ov_transform_points *Transform points from image coordinates to court coordinates or vice-versa*

Description

The court coordinate system is that used in `datavolley::dv_court()`, `datavolley::ggcourt()`, and related functions. Try `plot(c(0, 4), c(0, 7), type = "n", asp = 1); datavolley::dv_court()` or `ggplot2::ggplot() + datavolley::ggcourt() + ggplot2::theme_bw()` for a visual depiction. Image coordinates are returned as normalized coordinates in the range `[0, 1]`. You may need to scale these by the width and height of the image, depending on how you are plotting things.

Usage

```
ov_transform_points(x, y, ref, direction = "to_court")
```

Arguments

`x`	numeric: input x points. `x` can also be a two-column data.frame or matrix
`y`	numeric: input y points
`ref`	data.frame: reference, as returned by `ov_get_court_ref()` or `ov_shiny_court_ref()`
`direction`	string: either "to_court" (to transform image coordinates to court coordinates) or "to_image" (the reverse)

Value

A two-column data.frame with transformed values

References

https://en.wikipedia.org/wiki/Camera_matrix. For general background see e.g. Ballard DH, Brown CM (1982) Computer Vision. Prentice-Hall, New Jersey

See Also

`ov_get_court_ref()`, `datavolley::dv_court()`, `datavolley::ggcourt()`

Examples

```
## the ref data for the example image
crt <- data.frame(image_x = c(0.05397063, 0.95402573, 0.75039756, 0.28921230),
                  image_y = c(0.02129301, 0.02294600, 0.52049712, 0.51884413),
                  court_x = c(0.5, 3.5, 3.5, 0.5),
                  court_y = c(0.5, 0.5, 6.5, 6.5))

## show the image
img <- jpeg::readJPEG(system.file("extdata/2019_03_01-KATS-BEDS-court.jpg",
                          package = "ovideo"))
plot(c(0, 1), c(0, 1), type = "n", axes = FALSE, xlab = "", ylab = "",
     asp = dim(img)[1]/dim(img)[2])
rasterImage(img, 0, 0, 1, 1)

## convert the ends of the 3m lines on court to image coordinates
check <- data.frame(x = c(0.5, 3.5, 0.5, 3.5),
                    y = c(2.5, 2.5, 4.5, 4.5))
ix <- ov_transform_points(check, ref = crt, direction = "to_image")

## and finally plot onto the image
points(ix$x, ix$y, pch = 21, bg = 4)
```

ov_video_control *Functions for controlling the video player*

Description

The video element and the controls provided by this function are javascript-based, and so are probably most useful in Shiny apps.

Usage

```
ov_video_control(what, ...)
```

Arguments

what	string: the command, currently one of: • "play" (note that this requires that the playlist has already been loaded) • "stop" • "pause" • "prev" • "next" • "jog" - move the video forward or backwards by a given number of seconds (pass this value as the ... argument) • "set_playback_rate" - set the playback rate: 1 = normal speed, 2 = double speed, etc
...	: parameters used by those commands. For version 2 of the video controller, ... must include controller_var = "my_controller_var"

Examples

```
## Not run:
  ov_video_control("jog", -1) ## rewind 1s
  ov_video_control("jog", 10) ## jump forwards 10s
  ov_video_control("set_playback_rate", 0.5) ## play at half speed

## End(Not run)
```

ov_video_extract_clip *Extract clip from video file*

Description

Requires that ffmpeg is available on your system path.

Usage

```
ov_video_extract_clip(
  video_file,
  outfile,
  start_time,
  duration,
  end_time,
  extra = NULL,
  debug = FALSE
)
```

Arguments

`video_file`	string: path to the input file
`outfile`	string: path to the output file. If missing, a temporary file (with extension .mp4) will be used
`start_time`	numeric: start time in seconds
`duration`	numeric: duration in seconds. If missing, will be calculated from start_time and end_time
`end_time`	numeric: end time in seconds. If missing, will be calculated from start_time and duration
`extra`	: additional parameters passed to ffmpeg, in the form c("param", "value", "param2", "value2")
`debug`	logical: if `TRUE`, echo the ffmpeg output to the console

Value

The path to the video clip file

`ov_video_frame` *Extract one or more specific frames from a video file*

Description

Extract one or more specific frames from a video file

Usage

```
ov_video_frame(
  video_file,
  t,
  n,
  format = "jpg",
  debug = FALSE,
  framerate,
  method = "auto"
)
```

Arguments

`video_file`	string: path to the video file
`t`	numeric: the times of the frames to extract (in seconds)
`n`	integer: the frame numbers of the frames to extract. Ignored if `t` is provided. Frame numbering is 1-based (first frame at t = 0 is frame n = 1)
`format`	string: "jpg" or "png"
`debug`	logical: if `TRUE`, echo the ffmpeg output to the console
`framerate`	numeric: the framerate of the video. If not supplied, it will be found using [av::av_video_info]
`method`	string: the method to use, either "ffmpeg", "av", or "auto". "ffmpeg" is faster than "av" but requires that ffmpeg is available on your system path. If `method` is "auto", "ffmpeg" will be used if available and "av" if not

Value

The paths to the frame image files

See Also

`ov_video_frames()`

Examples

```
video_file <- ov_example_video(1)
img <- ov_video_frame(video_file, t = 5)
img <- ov_video_frame(video_file, n = 150)
```

`ov_video_frames` *Extract multiple consecutive frames from a video file*

Description

Requires that ffmpeg is available on your system path.

Usage

```
ov_video_frames(
  video_file,
  start_time,
  duration,
  end_time,
  outdir,
  fps,
  format = "jpg",
  jpg_quality = 1,
  extra = NULL,
  debug = FALSE,
  exec_fun
)
```

Arguments

`video_file`	string: path to the video file
`start_time`	numeric: start time in seconds
`duration`	numeric: duration in seconds. If missing, will be calculated from start_time and end_time
`end_time`	numeric: end time in seconds. If missing, will be calculated from start_time and duration
`outdir`	string: path to the output directory, which must exist. If missing, a temporary directory will be used
`fps`	numeric: frames per second, default is to extract all frames
`format`	string: "jpg" or "png"
`jpg_quality`	numeric: jpg quality from 1-31, lower is better (this is passed to ffmpeg as the `-qscale:v` parameter)
`extra`	: additional parameters passed to ffmpeg, in the form c("param", "value", "param2", "value2")
`debug`	logical: if `TRUE`, echo the ffmpeg output to the console
`exec_fun`	string or function: the function (or function name as a string) to use to execute the ffmpeg command. Defaults to `sys::exec_internal()`, or `sys::exec_wait()` if debug is `TRUE`

Value

If `exec_fun` has not been specified, the function will wait for the ffmpeg call to complete and then return a character vector of file names, one per frame. If `exec_fun` has been specified, the result of that function call will be returned immediately (because it might be a call to a background process)

See Also

`ov_video_frame()`

`ov_video_js` *Inject javascript for an HTML video player*

Description

Inject javascript for an HTML video player

Usage

```
ov_video_js(youtube = FALSE, twitch = FALSE, version = 1)
```

Arguments

`youtube`	logical: set to `TRUE` to include the Youtube API javascript. This isn't necessary if you are only using local video files
`twitch`	logical: set to `TRUE` to include the Twitch API javascript. Only with `version = 2`. Not necessary if you are only using local video files
`version`	numeric: code version. Default = 1, experimental = 2

Value

A head tag containing script tags

See Also

`ov_video_playlist()`

`ov_video_player` *Video player tag element*

Description

Video player tag element

Usage

```
ov_video_player(
  id,
  type,
  controls = FALSE,
  version = 1,
  controller_var = paste0(id, "_controller"),
  with_js = FALSE,
  ...
)
```

Arguments

`id`	string: the id of the tag
`type`	string: either "youtube", "twitch" (only with `version = 2`), or "local"
`controls`	logical: if `TRUE`, add "previous", "next", "pause", "stop", and "fullscreen" buttons. If `controls` is an object of class `shiny.tag` (created by `htmltools::tags()`) or `shiny.tag.list` (`htmltools::tagList()`) then those controls will added with this tag or tag list appended

`version`	numeric: code version. Default = 1, sort-of-experimental = 2. Version 2 supports multiple players on a single page, as well as `type = "twitch"`
`controller_var`	string: (for version 2 only) the js variable name to use for the controller object that controls this video player
`with_js`	logical: if `TRUE`, also include the supporting javascript libraries. If `with_js = FALSE`, you must make a separate call to `ov_video_js()` (e.g. in your Shiny ui.R function)
`...`	: other attributes of the player element (passed to the player `tags$div` call for youtube/twitch or `tags$video` for local)

Value

HTML tags. The outermost element is a div with id `paste0(id, "_container")`, with the player and optionally buttons nested within it.

Examples

```
## Not run:
  library(shiny)

  ## hand-crafted playlist for this example
  playlist <- data.frame(video_src = "NisDpPFPQwU",
                         start_time = c(589, 1036, 1163, 2731, 4594),
                         duration = 8,
                         type = "youtube")

  shinyApp(
      ui = fluidPage(
          ov_video_js(youtube = TRUE, version = 2),
          ov_video_player(id = "yt_player", type = "youtube",
                          version = 2, controller_var = "my_dv",
                          style = "height: 480px; background-color: black;",
                          controls = tags$button("Go",
                                 onclick = ov_playlist_as_onclick(playlist, "yt_player",
                                                         controller_var = "my_dv")))
      ),
      server = function(input, output) {},
  )

## End(Not run)
```

`ov_video_playlist` *Create video playlist*

Description

Create video playlist

Usage

```
ov_video_playlist(
  x,
  meta,
  type = NULL,
  timing = ov_video_timing(),
  extra_cols = NULL,
  normalize_paths = TRUE
)
```

Arguments

`x`	data.frame: a datavolleyplays object. Normally this will be a selected subset of the `plays` component of a datavolley object (i.e. a selected set of actions that you want the video playlist to contain)
`meta`	list: either the `meta` component of a datavolley object, or a list of such objects, or a data.frame with the columns "match_id" and "video_src". Entries in `video_src` should be paths or URLs to the video file associated with the corresponding `match_id`
`type`	string: currently "youtube", "twitch", or "local". If `type` is not specified as a parameter, and `meta` is a data.frame, then `type` can be provided as a column in `meta`. Alternatively, if `meta` is a `meta` component of a datavolley object, or a list of such objects, then `type` will be assumed to be "local". Note that a single playlist can't mix types, all entries must be of the same type
`timing`	list: the relative timing for each skill type, either a named list as returned by `ov_video_timing()` or a data.frame as returned by `ov_video_timing_df()`. See `ov_video_timing()` for further details
`extra_cols`	character: names of additional columns from `x` to include in the returned data frame
`normalize_paths`	logical: if `TRUE`, apply `normalizePath` to local file paths. This will e.g. expand the tilde in paths like "~/path/to/video.mp4"

Value

A data.frame with columns `src`, `start_time`, `duration`, plus any extras specified in `extra_cols`

Examples

```
## read data file
x <- datavolley::dv_read(datavolley::dv_example_file())
## note that this data file has no video specified, so put a dummy value in
dv_meta_video(x) <- "c:\\my_video.mp4"

## extract play-by-play data
px <- datavolley::plays(x)
## and put dummy video_time values in, because those are missing too
px$video_time <- sample.int(2e3, size = nrow(px))
```

```
## find pipe (XP) attacks in transition
px <- px[which(px$attack_code == "XP" & px$phase == "Transition"), ]

## create playlist
ply <- ov_video_playlist(px, x$meta, timing = ov_video_timing())

## with custom timing
ply <- ov_video_playlist(px, x$meta,
  timing = ov_video_timing_df(data.frame(skill = "Attack", phase = "Transition",
                          start_offset = -5, duration = 10, stringsAsFactors = FALSE)))
```

`ov_video_playlist_pid` *Create video playlist per point_id*

Description

Create video playlist per point_id

Usage

```
ov_video_playlist_pid(
  x,
  meta,
  type = NULL,
  extra_cols = NULL,
  normalize_paths = TRUE
)
```

Arguments

`x`	data.frame: a datavolleyplays object. Normally this will be a selected subset of the `plays` component of a datavolley object (i.e. a selected set of actions that you want the video playlist to contain)
`meta`	list: either the `meta` component of a datavolley object, or a list of such objects, or a data.frame with the columns "match_id" and "video_src". Entries in `video_src` should be paths or URLs to the video file associated with the corresponding `match_id`
`type`	string: currently "youtube", "twitch", or "local". If `type` is not specified as a parameter, and `meta` is a data.frame, then `type` can be provided as a column in `meta`. Alternatively, if `meta` is a `meta` component of a datavolley object, or a list of such objects, then `type` will be assumed to be "local"
`extra_cols`	character: names of additional columns from `x` to include in the returned data frame
`normalize_paths`	logical: if `TRUE, apply` normalizePath‘ to local file paths. This will e.g. expand the tilde in paths like "~/path/to/video.mp4"

Value

A data.frame with columns `src`, `start_time`, `duration`, plus any extras specified in `extra_cols`

`ov_video_timing` *Timing to use when creating video playlist*

Description

By default, all skills except reception have a timing of `c(-5, 3)`, meaning that the video clip will start 5 seconds before the recorded time of the event and end 3 seconds after its recorded time. Reception has a timing of `c(-2, 6)` (because reception usually has the same timestamp as the serve)

Usage

```
ov_video_timing(...)

ov_video_timing_df(x)
```

Arguments

`...`	: named parameters that will override the defaults. Each parameter should be a two-element numeric vector
`x`	data.frame: a data.frame of timings that will override the defaults, with columns `skill`, `phase`, `start_offset` (start offset in seconds, default = -5), and `duration` (duration in seconds, default = 8)

Details

`ov_video_timing_df` accepts and returns a data.frame rather than a named list. The data.frame format also allows timings to be differentiated by play phase ("Reception" vs "Transition").

Value

For `ov_video_timing` a named list, with names corresponding to skills ("Serve", "Reception", etc). For `ov_video_timing_df`, a data.frame with columns `skill`, `phase`, `start_offset`, and `duration`

See Also

`ov_video_playlist()`

Examples

```
## defaults
ov_video_timing()

## with different settings for serve and reception
ov_video_timing(serve = c(-2, 2), reception = c(-3, 1))

## as data.frame
ov_video_timing_df(data.frame(skill = "Set", phase = "Transition",
                              start_offset = -5, duration = 10))
```

Index

av::av_encode_video(), *16*
av::av_video_info, *32*

browseFile, 2

datavolley::dv_court(), *12, 13*, *28, 29*
datavolley::dv_read(), *3*
datavolley::ggcourt(), *12, 13*, *28, 29*
dv_meta_video, 3
dv_meta_video<- *(*dv_meta_video*)*, 3

editry::er_clip(), *9*
editry::er_clip_pause(), *9*
editry::er_clip_title2(), *9*
editry::er_clip_title_background(), *9*
editry::er_install_editly(), *8*
editry::er_layer(), *9*
editry::er_layer_news_title(), *9*
editry::er_spec(), *9*

htmltools::tagList(), *34*
htmltools::tags(), *34*

ov_3dpos_multicamera, 4
ov_cmat_apply, 6
ov_cmat_apply(), *7*
ov_cmat_estimate, 7
ov_cmat_estimate(), *4*, *6*
ov_editry_clips, 8
ov_example_video, 11
ov_ffmpeg_exe, 11
ov_ffmpeg_ok *(*ov_ffmpeg_exe*)*, 11
ov_find_video_file, 12
ov_get_court_ref, 12, *18*
ov_get_court_ref(), *29*
ov_get_video_data, 13
ov_get_video_data(), *25*
ov_get_video_meta, 14
ov_get_video_meta(), *27*
ov_images_to_video, 15
ov_install_ffmpeg, 16
ov_install_ffmpeg(), *11*
ov_merge_video_timing_df, 17
ov_overlay_data, 17, *18*
ov_overlay_image, *18*, 18
ov_playlist_as_onclick, 19
ov_playlist_to_html, 20
ov_playlist_to_html(), *24*
ov_playlist_to_video, 22
ov_playlist_to_vlc, 24
ov_playlist_to_vlc(), *21*
ov_set_video_data, 25
ov_set_video_data(), *14*
ov_set_video_meta, 26
ov_set_video_meta(), *14*, *25*
ov_shiny_court_ref, 27
ov_shiny_court_ref(), *25*, *28, 29*
ov_transform_points, *18*, 28
ov_transform_points(), *12, 13*
ov_video_control, 30
ov_video_extract_clip, 30
ov_video_frame, 31
ov_video_frame(), *33*
ov_video_frames, 32
ov_video_frames(), *32*
ov_video_js, 33
ov_video_js(), *35*
ov_video_player, 34
ov_video_playlist, 35
ov_video_playlist(), *21*, *23, 24*, *34*, *38*
ov_video_playlist_pid, 37
ov_video_timing, 38
ov_video_timing(), *36*
ov_video_timing_df *(*ov_video_timing*)*, 38
ov_video_timing_df(), *17*, *36*
ovideo, 4
ovideo-package *(*ovideo*)*, 4

sys::exec_internal(), *33*
sys::exec_wait(), *33*

utils::browseURL(), *3*