

# find frame in video

2023 way to do this:

```
ffmpeg -i "video.mp4" -r 1 -loop 1 -i image.png -an -filter_complex  
"blend=difference:shortest=1,blackframe=90:32" -f null -
```

Notice the 90 there, if set to 100 it would only match equally, with lower numbers you get to set a fuzzy search criteria

Alternative bash script from 2019:

[framepos.sh](#)

```
#!/bin/bash

# function framepos()

# search image in video
# 1. extract frames from video file
# 2. wait for N frames
# 3. search pattern in N frames
# 4. return frame position

# why step 2?
# cos calling findimagedupes is slow
# and buffering makes search much faster by factor 8

# dependencies:
# 1. ffmpeg aka avconv
# 2. findimagedupes, with "old school" perl backend

# author: milahu
# license: public domain OR creative commons zero OR mit license
# warranty: none. never trust my code
# version: 2019-05-14

# usage:
#
# see sample code at end of this file
#
# if you want to detect clips like intros and outros
# you must find "good key frames"
# who have strong contrast to other frames
# [ideally no animation]
# and who are encoded similar across many files,
# so you can use a lower threshold value.
```

```
#
# from these key frames,
# you can calculate cutting positions,
# by manual "edge detection" [video and audio]
# or you can run blackdetect or silencedetect filters
# near the key frames.
#
# to make search faster,
# limit your "search space" to where you expect a match.

# todo:
# * loop seems to hang in some cases [done?]
# * search backward, from end of video or from position
# * replace findimagedupes with imagemagick?
# * get all matches
# * rewrite in python
# --> use openCV?
# --> integrate into pyLoad?
#
https://realpython.com/fingerprinting-images-for-near-duplicate-detection/#what-libraries-will-we-need
# * use "frame exact" positions, avoid rounding errors [done]
# * round float seconds to %.3f ?
# * rewrite time2sec (etc) functions as "pipeline filters"
# so we can write: s=$(echo 01:23:45.678 | time2sec)
# * make calls to ffprobe more robust
# for example dont grep in file comments
# * add audiodetect, to search for audio clips
# similar to "voice/speech recognition"
# problem: search too fuzzy? library overkill?
# and: we need offset values = timestamps
# https://stackoverflow.com/questions/47121644
# audio-file-speech-recognition-in-python-location-of-word-in-seconds
# https://github.com/mozilla/DeepSpeech
# https://github.com/Uberi/speech\_recognition
# https://realpython.com/python-speech-recognition/
# simple:
# https://github.com/worldveil/dejavu
# https://github.com/tyiannak/pyAudioAnalysis/wiki/5.-Segmentation

# context:
# https://superuser.com/questions/663731
# find-video-file-position-using-an-image-match
# https://video.stackexchange.com/questions/11980
# is-there-a-program-for-automatic-clip-creation
# from-scene-detection-recognition
```

```
# https://github.com/Breakthrough/PySceneDetect
# python + openCV
# ffmpeg scene detect
# ffmpeg -i "$in" -filter:v "select='gt(scene,0.1)',showinfo" -f null
-
# https://superuser.com/questions/378639
# automatic-scene-detection
# kdenlive: scene recognition

# settings

# how much "similar" should matches be?
#threshold=90%
threshold=85%

# how long to wait for next extraction-progress check
# if your memory is limited, use smaller steps
step_duration=0.5
#step_duration=1

# compare at least N frames at once
# if your memory is limited, use smaller buffer
# average frame size: 6 MB for 1080p, 3 MB for 720p
frames_bufsize=100

#ffmpeg_cmd=avconv
ffmpeg_cmd=ffmpeg
ffprobe_cmd=ffprobe

# prefix for temp files. faster on a tmpfs filesystem
tmp_pre=/tmp/mv-$(date +%s).

# directory to save video files
# terminate with slash, or leave empty
v_out_prefix="video_out/"

# framepos()
# search images in video, print first position
# to search a "still image range"
# similar to the "blackdetect" filter in ffmpeg
# set $4 to 1
# and read "$time_first_frame $time_last_frame"
# sample:
# IFS=' ' read t1 t2 <<(framepos "$vid" "$img" 0 0 1)

function framepos() {
```

```

V="$1" # video file
T1="{2-0}" # time from
T2="{3-0}" # time to

R="{4-0}" # match still image range? 0 or 1

# image files
ARGV=("$@")
I="{ARGV[@]:4}" # image files = $5 $6 $7 ...

#echo "V = $V" >&2

# get frame rate
fps=$(get_fps "$V")

echo fps = $fps >&2

dur=$(time2sec $(get_dur "$V"))

echo "dur = $dur" >&2

echo near T1 = $T1 >&2
echo near T2 = $T2 >&2

# get exact frame times
# T_exact = round(T_near * fps) / fps
# round(F) = int(F + 0.5)
[[ "$T1" != '0' ]] && {
    if [[ "${T1:0:1}" = "-" ]]
    then
        #echo "got negative t1 $T1" >&2
        T1=$(echo $T1 $fps $dur \
            | awk '{printf "%.4f\n", int(($3 + $1) * $2 + 0.5) /
$2}')
    else
        T1=$(echo $T1 $fps \
            | awk '{printf "%.4f\n", int($1 * $2 + 0.5) / $2}')
    fi
    echo exact T1 = $T1 >&2
}
[[ "$T2" != '0' ]] && {
    if [[ "${T2:0:1}" = "-" ]]
    then
        #echo "got negative t2 $T2" >&2
        T2=$(echo $T2 $fps $dur \
            | awk '{printf "%.4f\n", int(($3 + $1) * $2 + 0.5) /
$2}')
    else
        T2=$(echo $T2 $fps \
            | awk '{printf "%.4f\n", int($1 * $2 + 0.5) / $2}')
    fi
}

```

```

    echo exact T2 = $T2 >&2
}

# extract frames. run in background
# bmp format is faster than png or jpg
echo extract frames to ${tmp_pre}frame-%04d.bmp >&2
ff_args=''
[[ "$T1" != '0' ]] && ff_args+=" -ss $T1 "
[[ "$T2" != '0' ]] && ff_args+=" -to $T2 "
# input seek = time arguments are before -i
$ffmpeg_cmd \
    $ff_args -i "$V" \
    ${tmp_pre}frame-%04d.bmp \
    2>/dev/null &
pid=$!

# print with custom delimiter
# assert that filenames dont contain delimiter
script_include=$(cat <<-'EOF'
    VIEW () {
        for f in "$@"
        do
            echo -n "$f"
            echo -ne "\t\t\t\t"
        done
        echo
    }
EOF
)

n2=0
found_first=0

#todo better?
t_last=-1
t=-2

doing_extract=false

while true
do

    n=$(ls ${tmp_pre}frame-*.bmp 2>/dev/null | wc -l)
    #echo found $n frames in "${tmp_pre}"'frame-*.bmp' >&2

    #echo n = $n >&2

    $doing_extract || {
        # $doing_extract is false
        echo -n "extract ... " >&2
        doing_extract=true
    }

```

```
}

(( $n == 0 )) && {
    kill -0 $pid 2>/dev/null || {
        # extract done
        echo debug found no match >&2
        break
    }

    kill -SIGCONT $pid
    sleep $step_duration
    continue
}

(( $n == 1 )) && {
    # only one frame extracted.
    # if ffmpeg is still extracting, this file is incomplete
    kill -0 $pid 2>/dev/null && {
        # extract running
        # last frame is incomplete --> $n-1 complete frames
        #echo extract continue with $(( $n - 1 )) frames >&2
        echo -n "$(( $n - 1 )) " >&2
        kill -SIGCONT $pid
        sleep $step_duration
        continue
    }

    n2=1
}

(( 1 < $n && $n <= $frames_bufsize )) && {
    # frame buffer not full
    # if extract is running, then wait before compare
    kill -0 $pid 2>/dev/null && {
        # extract running
        echo -n "$(( $n - 1 )) " >&2
        #echo extract continue with $(( $n - 1 )) frames >&2
        kill -SIGCONT $pid
        sleep $step_duration
        continue
    }

    n2=$(( $n - 1 ))
}

(( $n > $frames_bufsize )) && {
    #echo found $n frames
    # pause frame extraction to save space
    # extract is faster than compare
    echo $(( $n - 1 )) >&2
    #echo extract pause with $(( $n - 1 )) frames >&2
    kill -SIGSTOP $pid

    n2=$(( $n - 1 ))
}
```

```

    echo compare $n2 frames from $(ls ${tmp_pre}frame-*.bmp | head
-n 1) >&2

#todo fail if $I is empty, or files not found

doing_extract=false

break_while=false
for I_cur in "${I[@]}"
do
    echo "search for $I_cur" >&2

    # we need the "real path" for findimagedupes
    pattern=$(readlink -f "$I_cur")

    # call findimagedupes
    # to find "visually similar images"
    res=$(
        ls ${tmp_pre}frame-*.bmp \
        | head -n $n2 \
        | xargs findimagedupes -t $threshold \
          -i "$script_include" "$pattern" \
        | grep -a "$pattern"
    )

    if [ ! -z "$res" ]
    then
        res=$(
            echo "$res" \
            | sed 's/\t\t\t\t\t/n/g' \
            | grep -v '^$' \
            | grep -v "$pattern" \
            | sort \
            | head -n 1 \
            | sed -E 's/^. *frame-(.*?)\.bmp$/\1/'
        )

        # get frame time
        # note: frame numbers start with 1

        # matching frame:
        #echo "compare res get t, fps is $fps" >&2
        #t=$(
        #    echo $T1 $res $fps \
        #    | awk '{printf "%.4f\n", $1 + ( ( $2 - 1 ) / $3 ) }'
        #)

        # frame minus one:
        t=$(

```

```
        echo $T1 $res $fps \  
        | awk '{printf "%.4f\n", $1 + ( ( $2 - 2 ) / $3 )  
}'  
    )  
  
    echo "match t = $t" >&2  
  
    # no range search  
    (( $R == 0 )) \  
&& {  
        # return  
        echo $t  
  
        # stop extracting  
        kill -9 $pid 2>/dev/null  
  
        # remove all temp files  
        rm ${tmp_pre}frame-*.bmp  
  
        break_while=true  
        break  
    } \  
    || {  
        # range search  
        (( $found_first == 0 )) \  
&& {  
            # range start  
            t_first=$t  
        }  
        found_first=1  
    }  
fi  
  
#todo range search might be broken  
# range search and  
# found first frame and  
# current frame is no match  
(( $R == 1 && $found_first == 1 && $t == $t_last )) \  
&& {  
    # t_last is last frame of still image range  
  
    # return  
    echo "$t_first $t_last"  
  
    # stop extracting  
    kill -9 $pid 2>/dev/null  
  
    # remove all temp files  
    rm ${tmp_pre}frame-*.bmp
```



```

        break_while=true
        break
    }

done

$break_while && break

#todo combine range search and multi image?
# only use first match as "range image"?

t_last=$t

# remove processed temp files
(( $n2 > 0 )) \
&& ls ${tmp_pre}frame-*.bmp | head -n $n2 | xargs rm
done
}

# convert HH:MM:SS.NNN to float seconds
function time2sec () {
    date -u -d "1970-01-01 $1" +%s.%3N
}

# convert float seconds to HH:MM:SS.NNN
function sec2time () {
    date -u -d "0+$1sec" +%T.%3N
}

# get video duration
function get_dur() {
    $ffprobe_cmd -i "$1" 2>&1 | grep Duration | cut -d' ' -f4 | cut -d,
-f1
}

# get video frame rate
function get_fps() {
    $ffprobe_cmd "$1" 2>&1 \
    | grep 'Video: ' \
    | head -n 1 \
    | sed -E 's/^.*(.*) fps.*$/\1/' \
    | sed 's/^23.98$/23.976/'
    # ffprobe prints wrong fps, change 23.98 to 23.976
}

# sample use

```

```
# video + image from argv
# call this script like: ./framepos.sh video.mp4 image.png
v="$1"
i="$2"

# search range. negative = seek from end
a=-12
b=-8

# get frame position
t=$(framepos "$v" $a $b 0 "$i")

# multi image search
#t=$(framepos "$v" $a $b 0 "$i_1" "$i_2" "$i_3")

if [ ! -z "$t" ]
then
    v_out="{v_out_prefix}${v%.*}.cut.${v##*}"
    echo "cut video to $v_out"

    # if you need absolute paths ...
    #script_dir=$(readlink -f "$0")
    #script_dir="{script_dir%/*}"

    # create dst directory
    if [[ ${v_out_prefix: -1} == "/" ]]
    then
        mkdir -p -v "$v_out_prefix"
    fi
    v_out_dir="{v_out%/*}"
    mkdir -p -v "$v_out_dir" 2>/dev/null

    # cut video
    # use '-map 0' to copy all streams
    ffmpeg_cmd -i "$v" -ss 0 -to "$t" -c copy -map 0 "$v_out"

else
    echo error frame not found
fi
```

<https://github.com/milahu/random/blob/master/framepos.sh>

<https://superuser.com/questions/663731/find-video-file-position-using-an-image-match>

From:  
<http://wuff.dyndns.org/> - **Wulf's Various Things**

Permanent link:  
<http://wuff.dyndns.org/doku.php?id=howto:video-framepos>

Last update: **2024/01/15 00:49**

