CUSTOM PYTHON SCRIPT FOR DV FILE ANALYSIS

Patrick Feaster
October 29, 2020

IUMDS has developed and used a custom Python script to carry out a frame-by-frame comparison of Memnon's simultaneously captured DV (FireWire) and SDI (MKV) transfers of DV tapes. This script helped us (and them) to troubleshoot our approach to reformatting before putting it into production, and it has since been a regular tool in our quality control for the format. The present document contains:

- *A narrative account of the script's origin, development, and findings*
- *The text of a public presentation about the script from October 2020*
- *Two versions of the script itself (one developed specially to run on a server from a command line)*
- *A copy of a report of findings presented to Memnon in February 2020*

The idea of creating a script to compare DV and SDI files first came up during a videoconference between staff from IUMDS and meemoo (formerly VIAA) on January 16, 2020, when Brecht Declercq brought up some questions about how the DV files Memnon was providing—often as multiple brief segments—related to the SDI files with a continuous time base that reflected the actual chronology of playback.

I'd previously been analyzing DV files on behalf of IUMDS with DVAnalyzer, and had written about one test on January 14 that had raised some similar questions:

> I've attached the DV Analyzer results from the DV test file. The main DV Analyzer screen says "No identified errors," as I mentioned at the standup. But the detailed report flags some spots in the file that warrant closer attention.
>
> The lines marked "S" appear just to correspond to points where there's a scene change. I believe it means "start," as in the recorder starting and stopping.
>
> The lines marked "N" indicate nonsequential timecode. These points *do* seem to correspond to glitches, at least some of the time. See the "jump" at 13:04, for example.
>
> I guess this doesn't count as an "error" for DV Analyzer's purposes, which is interesting -- maybe the program uses "error" only to refer to problems caught by checking parity data?
>
> I need to re-download the MKV file -- my computer ran out of space the first time I tried to download it -- but I'm curious whether the same glitches appear in the SDI.

My idea, as of January 16, was to write a script that would compare the files frame by frame so that we could see whether anomalies in the metadata had arisen at the immediate point of playback (and so would be reflected in the SDI too) or whether they were due to some issue with the capture of the DV stream after it had left the deck.

I was more comfortable coding in MATLAB at that point, but I soon discovered that the MATLAB video reader wasn't able to read in frames from files in the DV format, so I switched to Python, which I knew less well but was then already being encouraged by MDPI leadership to learn as an alternative. On January 17, I reported:

> I've figured out how to extract frames from the DV and MKV files using OpenCV in Python, as a first step towards comparing the file contents. The first thing I notice is that the frames are different sizes:
>
> DV (FireWire): 720x480
>
> MKV (SDI): 720x486
>
> MOV (SDI mezzanine): 720x512
>
> It looks like the MKV and MOV are just padded out with black pixels, so I should be able to crop them to the same height as the DV frames for analysis. But I wanted to make sure this is expected behavior otherwise.

Rob Mobley confirmed that these were the expected frame sizes, so I proceeded to work out how to crop the larger frames so that they would precisely match the dimensions of the DV frames. For the MKV files, this meant excerpting rows 4 through 484. For the MOV files, it meant excerpting rows 29 through 509.

I now found through experiment that "matching" frames weren't digitally identical, which meant that I couldn't simply check to see whether the pixel values were exactly the same – they wouldn't be. So I turned instead to the structural similarity index metric from the Scikit-Image library.[1] Tests performed on MKV files showed that matching frames typically scored above 99%, while non-matching frames typically scored below 90%, even if the difference between them was barely perceptible to the eye. During this early period, I experimented with different threshold values (99%, 98%, etc.) for a "match." However, I also found that I could achieve reliable results this way only by analyzing the preservation (DV) and preservation intermediate (MKV) files. The mezzanine (MOV) and access (MP4) copies had too many artifacts of compression for the structural similarity analysis to judge subtle differences between frames accurately.

On January 22, I wrote:

> I have a Python script working now that carries out a frame-by-frame comparison of a DV (FireWire) and SDI (MKV) file. It assumes that the DV file might drop frames but that the SDI file won't, which seems to be borne out by experiments so far.
>
> The results from the first group of test files are interesting.

[1] This function was initially **measure.compare_ssim**, with **measure** imported from **skimage**, but when I updated to a new version of **skimage** sometime in mid-2020 (together with some other updates) it had been relocated to **metrics.structural_similarity**, with **metrics** imported instead of **measure**.

First, the DV file starts 20 frames before the SDI file does, during the color bars. Frame 21 in the DV file corresponds to frame 1 in the SDI file. The twenty color-bar frames missing from the SDI file are "real" frames with timecode and might contain audio -- I haven't checked.

[Note: it was the need to deal with these color bars in the first test tape that led me to begin each analysis by seeking the first *non-identical* frame in the DV file to try to match to a frame in the SDI file. Otherwise, any color-bar frame would have matched pretty much any other color-bar frame. My goal was to find the first frame that differed from a preceding frame and to assess "matches" backwards and forwards from that. This was helpful at a point when we weren't sure whether the DV and SDI files were necessarily starting at the same time. It became less helpful later on, once we'd established this point to our satisfaction, and is now something of a "vestigial" part of the code.]

Second, every "N" entry in the DVAnalyzer report -- indicating nonconsecutive timecode -
- corresponds to one or more frames that are missing from the DV file but present in the MKV. The number of dropped frames varies quite a lot. This isn't obvious from the DVAnalyzer report, which lists the nonconsecutive timecode readings but doesn't indicate how far off each one is from the expected reading -- you'd have to do some inconvenient calculations to work this out. And as came up in our conversation with VIAA, we didn't know for sure whether the problem was with actual dropped frames or with the timecode. Now we know the timecode is correct, or at least that discrepancies in timecode represent real glitches.

Here's a partial report (I stopped analysis partway through):

> Frame 23499 in DV doesn't match frame 23479 in SDI
> Dropped frames: 40
Frame 23499 in DV corresponds to frame 23519 in SDI
> Frame 23500 in DV doesn't match frame 23520 in SDI
> Dropped frames: 1
Frame 23500 in DV corresponds to frame 23521 in SDI
> Frame 23501 in DV doesn't match frame 23522 in SDI
> Dropped frames: 22
Frame 23501 in DV corresponds to frame 23544 in SDI
> Frame 23502 in DV doesn't match frame 23545 in SDI
> Dropped frames: 2
Frame 23502 in DV corresponds to frame 23547 in SDI
> Frame 23616 in DV doesn't match frame 23661 in SDI
> Dropped frames: 12
Frame 23616 in DV corresponds to frame 23673 in SDI
> Frame 23645 in DV doesn't match frame 23702 in SDI
> Dropped frames: 18
Frame 23645 in DV corresponds to frame 23720 in SDI
> Frame 23699 in DV doesn't match frame 23774 in SDI
> Dropped frames: 68
Frame 23699 in DV corresponds to frame 23842 in SDI
> Frame 23709 in DV doesn't match frame 23852 in SDI
> Dropped frames: 1
Frame 23709 in DV corresponds to frame 23853 in SDI

> Frame 23710 in DV doesn't match frame 23854 in SDI
> Dropped frames: 1
Frame 23710 in DV corresponds to frame 23855 in SDI
> Frame 23889 in DV doesn't match frame 24034 in SDI
> Dropped frames: 29
Frame 23889 in DV corresponds to frame 24063 in SDI
> Frame 24835 in DV doesn't match frame 25009 in SDI
> Dropped frames: 352
Frame 24835 in DV corresponds to frame 25361 in SDI
> Frame 24836 in DV doesn't match frame 25362 in SDI
> Dropped frames: 7
Frame 24836 in DV corresponds to frame 25369 in SDI
> Frame 27188 in DV doesn't match frame 27721 in SDI
> Dropped frames: 5
Frame 27188 in DV corresponds to frame 27726 in SDI

As you can see, sometimes there's just a single dropped frame, but there can be a lot more -- in one case, there are 352 of them in a row. Do we understand why these frames are making it into the SDI file but not the DV file?

Another conclusion is that the DVAnalyzer report is catching all these glitches, even if it doesn't tell us clearly how big a glitch we're dealing with in each case. Every time my script found dropped frames, the DVAnalyzer report also shows a timecode problem. So I'm going to try writing a revised version of the Python script that only looks at frames flagged by DVAnalyzer, to speed things up. Right now the script takes about five minutes to analyze every minute of video, which is frustratingly slow.

I should also be able to revise the script to compare one MKV file with a set of DV files representing multiple segments (plus a DVAnalyzer XML export for each segment), but I haven't done that yet.

I did write—and begin using—a revised version of the script that looked only at frames flagged as anomalous by DVAnalyzer (I had to figure out how to import the relevant frame numbers from DVAnalyzer XML, which I did using **ElementTree**) in order to speed up processing, although this later turned out to have been a step in the wrong direction. The next day (Jan. 23), after running a complete analysis on the tape that reaffirmed the earlier finding, I wrote:

I know we're expecting DV to be glitchy, but the fact that all these frames are in the SDI file but not the FireWire file points to some problem with the capture setup for the DV stream. There wasn't any issue with reading the DV tape, apparently, but only with getting the FireWire output onto the drive. At one point it looks like the DV file skips 11.745 seconds where there's no problem at all in the SDI. Rob and I are going to take a closer look at the video files themselves to confirm -- I can't monitor them very well on my own workstation. There's a lot of discussion of dropped frames from FireWire DV transfers on prosumer forums, with people blaming factors such as insufficient drive speeds and wrongly configured capture software.

Mike Casey replied:

> Isn’t this expected behavior? The DV tape has playback issues, drops data, while the SDI uses error correction and whatever else to recover and use the data?

To which I responded:

> Unfortunately, no, this doesn't appear to be expected behavior, or at least it's not something Rob or I would have expected to see based on our understanding of DV and its problems.
>
> If the DV tape had playback issues and dropped data so severely that it wasn't possible to capture a single frame over FireWire for eleven seconds (but without the software starting a new segment?!) we'd expect to see SDI frames that are held ("frozen") or garbled. In this case, the SDI frames are fine, and contain "real" content played successfully from the tape. We're evidently dealing with content read successfully from the tape but not written, for some unknown reason, to the DV file.
>
> In cases of interpolation, the SDI file and the DV file should still contain the same frames, the difference being that the DV file also contains metadata about the error concealment.
>
> Segmenting is a different issue, where problems reading metadata cause the stream-capture software to think there's been a change of specs. But even then there probably shouldn't be a case where the SDI file contains a frame that isn't in any of the DV files.
>
> This is much more the behavior we'd expect to see from a drive not being fast enough to write the data being sent to it, or from something keeping data from passing correctly over the FireWire after it's been read.
>
> That would also be consistent with the fact that this particular tape displays no other problems whatsoever. According to DVAnalyzer, there wasn't a single video error concealment throughout. There's no sign the *player* had any trouble with it. But if there's a problem with the drive or FireWire, it would affect all tapes equally, even tapes without glitches of their own.
>
> It's possible people have run into similar problems before and misdiagnosed them because they had no way of knowing what the player had "really" read from the tape (as we do thanks to the simultaneous SDI file). If people have assumed the fault lies with glitchy DV tapes, they might not have tried troubleshooting other links in the chain.
>
> Or there could be some other explanation. But if so, we don't know what it is.

Memnon re-digitized the tape we’d analyzed before, and I analyzed both the new DV file and the old DV file with reference to the new SDI file.

> I've attached two reports for comparison: "transfer_1.txt" is a report on the first transfer of the test tape, and "transfer_2.txt" is a report on the second, more recent transfer. Both comparisons were made with the new SDI transfer so that the SDI frame references will be consistent. As you'll see, I've expanded the report to include all the various "event" and "flag" types indicated by DVAnalyzer **[imported using the same methods I’d been using to import “problem” frame numbers]**, and I've also added times (hh:mm:ss) to help with locating points of interest in the video files. **[Up until that point I’d been calculating times by hand, dividing the frame number by the frames per second, which was a nuisance, and something much more efficiently handled by programming.]**

> The new transfer has fewer dropped frames -- just 42, as compared to 563 last time -- but the fact that it has any at all should still be a concern. The dropped frames are also in different places than before, which suggests that they don't represent problems with the tape itself. Adam and I confirmed that one of the larger dropped sections is visible as a glitch in the new DV file.
>
> Meanwhile, the new transfer contains 886 "DV timecode incoherencies," which didn't come up at all in the previous transfer. I'm not sure what a "DV timecode incoherency" is, but I'm looking through DVAnalyzer documentation to see if there are any clues. Whatever they are, they weren't an issue before, but now we're getting lots and lots of them.
>
> The new transfer has a concealment of 27 video errors at DV frame 10376 that wasn't in the earlier transfer.
>
> The "recording started" metadata markers are offset by one frame relative to the SDI transfer for a section of the tape, with the old file pointing to SDI frames 19554, 19844, 20189, 20508, 21358, and 23412 and the new file pointing to SDI frames 19555, 19845, 20190, 20509, 21359, and 23413, i.e., consistently one frame ahead. I don't know which file is "correct," if either; I just know that the time references don't match. The markers line up in the two files both before and after this point (at 18979 and 24651).

I reported these findings to Memnon at the operations meeting of January 28, 2020. We also had a follow-up meeting on January 30 with Brecht Declercq and his team, at which point we described my script and what we'd found with it. I then sent them a copy of the script for their own use, and they obtained similar results with it, noting in an email of February 13: "we had one case in which the .dv version lacked ca. 7.5 seconds as compared to the .mkv version."

On the afternoon of January 30, I analyzed a third Memnon transfer of the same tape, consisting this time of three segments, and now submitted along with Memnon's own DVAnalyzer results; and reported:

> There aren't any great surprises -- the DVAnalyzer reports generated by Memnon are identical to the ones I generated myself. I used the second SDI transfer as a reference again for the attached report, to make it easier to compare with the previous two reports. I compared the new FireWire transfer against the new SDI transfer too, though, and there weren't any significant differences.
>
> The number of dropped frames has gone down again, and this time there are only four -- two pairs of two.
>
> There aren't any of the "incoherencies" we saw with the previous set of files, so that problem seems to have gone away.
>
> There's one minor error concealment, at a different point in the tape than we saw last time. This behavior is probably to be expected from the format.
>
> I'm not sure what to make of the "repeating arbitrary bit" at DV frame 23410, which we haven't seen before, but it doesn't seem particularly worrisome -- it coincides with a spot where recording started and stopped.

The "recording start" markers are different yet again from the other two transfers. The markers in the three transfers point to the following SDI frames:

18979 / 18979 / 18979
19554 / 19555 / **19554**
19844 / 19845 / **19844**
20189 / 20190 / **20189**
20508 / 20509 / **20508**
21358 / **21359** / **21359**
23412 / **23413** / **23413**
24651 / 24651 / 24651

I suspect "recording start" markers might just not align exactly with frames, as recorded on the tape, so that the frames they end up associated with in the DV file can legitimately differ between different plays.

Segment one seems to contain all the content we'd expect from the earlier transfers. Segments two and three show the concluding color bars, without any DV timecode read from the tape for them. Segment two contains just a single frame (which answers our question about whether that's possible), and segment two contains fourteen frames. There's no difference in technical metadata associated with the segments.

At this point, we were probably exposing discrepancies between the two files that reflected subtleties of performance nobody had previously been in a position to detect. Memnon now proceeded to reformat some additional tapes in the interests of assessing a greater variety of material. I wrote on February 4:

I've attached reports for the five new sets of transfers made using the Python comparison script. I've had to tweak the script a bit today -- lowering the threshold slightly for what counts as a matching frame **[from this point on I settled on 96% as the lower limit for a "match"; this has seemed to achieve the best balance in practice between false positives and false negatives as applied to a wide range of tapes]**, as well as sometimes manually identifying an initial pair of matching frames. With those adaptations, I think it's still working, although I haven't gone through frame by frame to confirm.

Anyhow, it looks like there are still dropped frames, but not very many, and almost always *single* dropped frames. One of the files (40000003823665) has lots of those "timecode incoherency" errors again. Another (40000003695725) has a huge amount of error concealment, but I guess that's to be expected.

The large segments that are missing from the DV files **[previously detected by Adam Nickel and Rob Mobley]** are a bigger concern, though, particularly in the case of 40000003823665, where there doesn't seem to be any obvious reason for it. **[At least some of this turned out to be due to difficulty in transitioning between LP and SP segments, which we ultimately accepted as unresolvable—I won't cover all of our experiments on that front, which included mocking up a "test" tape of our own with known properties.]**

I reported these findings at another operations meeting on February 5. On February 6, I wrote to John Macdonald:

> Here are the quantities of dropped frames my script found, together with the "clues" in the DVAnalyzer reports:
>
> 40000000329112= 1, noted as "non-consecutive recdate/rectime" and a recording start point at frame 68155
> 40000003819002 = 5, at each of the points noted "non-consecutive DV timecode"
> 40000003695725 = 11, mostly at points noted "non-consecutive DV timecode"
> 40000003823665 = 1, at the point noted "non-consecutive DV timecode"
> 40000003474808 = 0
>
> The last two are also missing longer segments, as we discussed -- this just covers small quantities of dropped frames.
>
> I've been going through to confirm these findings through a manual frame-by-frame comparison and am not quite done yet, but so far the quantities of dropped frames seem to be right. One other thing I've done has been to revise the script so that it will export the frames it thinks don't match, which can help confirm what's going on.

As mentioned here, I'd begun having the script export still images of side-by-side DV and SDI frames under certain circumstances (when they failed to match, when they did match after having previously failed to match, and so forth). I'd been finding it frustrating to locate and compare individual frames by hand. But as I carried out more of these analyses, I began to have second thoughts about restricting analysis to frames that had been flagged as anomalous by DVAnalyzer, as I wrote on February 11:

> I've been doing some further analysis of the DV test files to make sure the numbers of dropped frames we've been reporting are accurate. The short answer is that it appears they are.
>
> The longer answer is that they are, but that they're not necessarily in places we'd expect based on DVAnalyzer reports.
>
> Comparing every frame in the DV and SDI files is time-consuming, so to get results more quickly, I'd modified the script to check only those frames in the DV files that DVAnalyzer had flagged as having issues. In practice, this meant that if the two files got out of sync, the script would discover and report the problem at the point of the next DVAnalyzer-reported error, even if the problem had actually occurred earlier in the sequence. We'd still get the right number of dropped frames in that scenario, but we wouldn't know their locations.
>
> With this last batch of tests, I noticed that problems seemed to be associated not just with reports of "nonsequential DV timecode," as we'd come to expect, but also with other reported issues such as video error concealments or repeating arbitrary bits. So I thought it would be worthwhile to re-analyze the files in the slower, more time-consuming way, checking every pair of frames, and to have the script export the frames it thought didn't match, for further examination. I also adjusted the script so that it could handle cases in which pairs of frames were different enough not to match, but without any loss of synchronization.

I re-analyzed 40000003695725 in this way -- that's the tape of the Hoagy Carmichael birthday celebration at the ATM, which had previously given us 81 separate segments.

The count of dropped frames is 11, the same as before. But many of these dropped frames appear in places where DVAnalyzer didn't report any problems, e.g., at frame 116788.

I thought there was a possibility that the error was in the SDI file instead of the DV file, so I went through and in each case exported the SDI frame *before* the one that was reported missing from the DV file, in case the SDI file was doubling up frames, or something like that. But that doesn't seem to be the case. See for example the five SDI frames in the zip folder 116796-116800. The middle (third) frame is missing from the DV file, but it looks (to me, at least) like a "real" frame rather than a glitch or interpolation. I'd welcome second (or third, or fourth) opinions.

This suggests that there can be dropped frames that don't register as DV timecode glitches.

Shortly after, we received a new set of test files created on February 12. On February 14, I wrote:

I've been analyzing the latest set of files. I'm using the slower but more robust/detailed method, so it's taking a while, but for 40000003823665 (the TAI instrument maker video) there are once again a number of single dropped frames, as the DVAnalyzer reports of nonconsecutive timecode would suggest.

For what it's worth, one program which online video forums associate with solving the dropped-frames problem is WinDV, a specialized (but free) tool designed exclusively for writing DV FireWire video streams to file as reliably as possible, with buffering to accommodate data hiccups:

http://www.windowsmoviemakers.net/PapaJohn/61/WinDV.aspx

While the documentation states that WinDV creates "AVI files," I think these may be effectively the same as our DV files, or easily converted into them by changing the header. Here's a discussion of type 1 and type 2 file structure options which implies that the output files are fairly straightforward stream captures:

https://support.corel.com/hc/en-us/articles/216275758-DV-File-Types-and-Pinnacle-Studio-Compatibility

I'd be reluctant to conclude that dropped frames are just something we have to live with unless we can confirm that WinDV -- or something equivalent -- wouldn't help. Memnon's setup with DirectShow may just not have a sufficient data buffer. A lot of all-purpose capture systems don't, which is probably why WinDV has such an enthusiastic following.

Meanwhile, 50000000025060, the newly-made test tape, has a longer segment dropped after the first recording stop point, which I think corresponds to a speed change. So it looks like the DV capture is still failing to accommodate changes in speed. It does pick up the stream again later -- probably after the original recording speed resumes -- but without creating a new segment. The current version of my script doesn't expect such long gaps, so I may need to tweak it before it can provide a more detailed look at what's going on.

Later same day, I added:

> Here's a report for the new transfer of 40000003823665, the TAI instrument-maker video.
>
> It reports fifteen dropped frames in all, and many of them are "real" and correspond to problems reported in DVAnalyzer, but this time there also seem to be some issues with the SDI file. DV frame 65629 should match SDI frame 65642, but SDI frame 65642 is garbled, with a piece from the right part of the frame relocated to the left part of the frame -- it's on the right in the attached image. Adam and I confirmed that the glitch is visible in the video, and it's in all three versions (pres, mezz, access). The same DV frame does match SDI frame 65643, however. So it looks as though SDI frame 65642 is garbled and spurious, and that SDI frame 65643 is what SDI frame 65642 should have been.
>
> I've seen a couple other cases of SDI frames garbled in this same way, but only the one case for this particular tape.
>
> SDI frames 63176 and 63177 both appear to be duplicates of DV frame 63164.
>
> So in addition to the DV file dropping occasional frames that are in the SDI file, the SDI file seems occasionally to be doubling up a frame, sometimes garbled.

And on February 18:

> Here's the significant part of the report on 40000003474808, the third and final file set in the latest DV results:
>
> DV 1:14634 (0:08:08) > expected SDI = 14635 (0:08:08); actual SDI = 14636 (0:08:08); dropped frames=1
> DV 1:36974 (0:20:34) > expected SDI = 36976 (0:20:34); actual SDI = 36977 (0:20:34); dropped frames=1
> DV 1:59309 (0:32:59) > expected SDI = 59312 (0:32:59); actual SDI = 59313 (0:32:59); dropped frames=1
> DV 1:81643 (0:45:24) > expected SDI = 81647 (0:45:24); actual SDI = 81648 (0:45:24); dropped frames=1
>
> None of these DV frames corresponds to a timecode anomaly reported by DVAnalyzer. I've been trying to decide whether SDI frames 14635, 36976, 59312, and 81647 are "real" frames or spurious frames, either repeating a previous frame or interpolated midway between the frames to either side. To do this, I've been extracting sets of three frames: the questionable frame plus the frames to either side of it. Results are attached in a zip folder for us to look at in our upcoming meeting.
>
> I've also attached the exported mismatched and resynchronized frames from 50000000025060, the test tape recorded in 224, although I think we can probably see what's going on with that tape just as well by looking directly at the video.

Later that same day, I reported our findings to Memnon at a third operations meeting, which I recall as being somewhat tendentious, with Andrew Dapuzzo looking at first to dismiss any anomalies as par for

the course with such a glitchy format, but ultimately agreeing to investigate further. Partly in response to this push-back, I prepared a more formal written report of our recent findings, "REPORT ON ANALYSIS OF DV TEST FILES FROM FEBRUARY 12, 2020," attached here as Appendix D and shared with John Macdonald and Ari Swartz on February 20.

Now that I was beginning to receive some sets of files in which we had one SDI file but multiple DV files, I'd revised the script to try to get it to cycle through the DV files and compare them each in sequence with the SDI file, with one beginning where the preceding one left off. I didn't yet have it working on February 20, when I wrote:

> I left an analysis of the 20200217 transfer of tape 40000003695725 running overnight. I need to do some work to get my script to handle transitions between separate DV segments better, but I got a solid set of results for the first DV segment and wanted to share them with you sooner rather than later.
>
> Here's an abridged report:
>
> SDI = F:/Tests 20200217/40000003695725_Sony/MDPI_40000003695725_01_presInt.mkv
> Similarity threshold = 0.96
> RESULTS >>> RUN AT 17:21:21
> DV = F:/Tests 20200217/40000003695725_Sony/MDPI_40000003695725_001_pres.dv
> ==========
> DV 1:3168 (0:01:46) > non-consecutive DV timecode/repeating arbitrary bit/
> DV 1:3168 (0:01:46) > expected SDI = 3169 (0:01:46); actual SDI = 3170 (0:01:46); dropped frames=1
> DV 1:19081 (0:10:37) > expected SDI = 19083 (0:10:37); actual SDI = 19084 (0:10:37); dropped frames=1
> DV 1:41595 (0:23:08) > expected SDI = 41598 (0:23:08); actual SDI = 41599 (0:23:08); dropped frames=1
> DV 1:64092 (0:35:39) > expected SDI = 64096 (0:35:39); actual SDI = 64097 (0:35:39); dropped frames=1
> DV 1:86572 (0:48:09) > expected SDI = 86577 (0:48:09); actual SDI = 86578 (0:48:09); dropped frames=1
> DV 1:109035 (1:00:38) > expected SDI = 109041 (1:00:38); actual SDI = 109042 (1:00:38); dropped frames=1
>
> The script reports discrepancies as "dropped frames," but of course they could also be "extra frames." **[I subsequently revised the script to call these "discrepant" frames, which I thought was a more neutral alternative.]**
>
> The first reported discrepancy corresponds to a DV timecode anomaly, but the other five don't and are regularly spaced, just as we saw in the 20200212 transfer of tape 40000003474808. In that case, the discrepancies came at intervals of 22340, 22335, and 22334 frames. This time, they're at intervals of 22514, 22497, 22480, and 22463 frames. In both cases, the interval is right around 12.5 minutes and decreases gradually over the course of the tape. Were both transfers made with the same equipment?

After tweaking the script a little further, I added on February 25:

> A quick follow-up. I've now got my script handling transitions between multiple FireWire segments (thanks to having more examples to experiment with), so I was able to analyze the second and third segments for this tape. There's one more "extra" frame at SDI frame 131495 (between DV frames 3:21516 and 3:21517, where "3" is the segment number). That's a 22453-sample gap, which fits the usual pattern.

That same day, February 25, I talked briefly with John Macdonald, who said Memnon had experimented with using WinDV and experienced no dropped frames. They were looking into how best to extract the relevant pieces of WinDV to incorporate into their system. About this time, Memnon also requested a copy of my script so that they could run it themselves, so on February 27, I shared an updated version with them as well as with meemoo.

On March 2, I analyzed three new transfers made with Memnon's revised system, reportedly incorporating some code borrowed from WinDV. One consequence of the changes they'd made was that from this point onward we received only a single DV segment per tape. We only began asking serious questions about this arrangement in October, when Brecht Declercq, Mike Casey, and I were preparing our presentation on DV transfer strategies for the IASA / FIAT-IFTA conference and wanted to make sure we were explaining it accurately. Apparently, Memnon's position was that creating a segment break at every change of specification, as meemoo had originally wanted, would have resulted in an unmanageably large number of segments. Instead, the agreement with meemoo, and by tacit extension (?) with IU, was that packets were henceforth to be written to the DV file "as is," without regard for changes in specification between them. This approach wouldn't technically violate the DV format, but discrepancies in specification would likely produce files that existing software couldn't play (as Brecht Declercq confirmed), since it assumes a constant specification. But Andrew Dapuzzo also indicated that with IU, differences in specification, such as a change in audio specification, were leading in practice to tapes being rejected, and that there had been about twenty such cases as of October 2020. Whatever the explanation, the bottom line is that from this point forward we stopped receiving multiple DV segments, so that the part of my script that was designed to handle this situation was no longer useful to us. Nor did we receive any files that seemed to contain incompatible parts with different specifications, although it's not clear how these were "caught."

In any case, this time the script found no dropped frames in any of the three file sets, although a common pattern now began to appear in which a number of DV frames duplicating a single SDI frame appear immediately after an end-of-recording flag. There was something of a wait over the next couple weeks. On March 11, I wrote:

> I just went down to check with John and Ari about where things stand with getting that new batch of DV files for us to analyze. They've run into a few new issues trying to get seven lines running simultaneously, and it sounds like Denis wants to review all files locally at Memnon before passing them along to us. So I'm not sure things are quite as far along as I understood they were from John when he came to pick up the external drive on Monday -- it had sounded as though they were ready to load the drive up immediately at that point. I've indicated that it would be helpful for us to get some of the files sooner rather than later, even if it's not yet a full set of seven, so that we could start making some headway on the analysis. I'll let you know as soon as something is ready.

At the end of March, IUMDS staff shifted to at-home work due to the COVID-19 pandemic. Just as operations were shutting down at the Innovation Center, I collected a portable drive containing a new

set of transfers from Memnon, installed or updated the necessary software on my personal laptop, and began DV analysis at home around March 25. There were also some efforts by Memnon to distribute files online via Ci with IUMDS staff as part of early steps towards confirming what a submission package would look like for this format.

On April 6, I wrote in response to another IUMDS staff member's observation that the audio on one tape "has intermittent distortion throughout on both .dv and .mkv, probably on original carrier":

> I never listened through to the whole thing, but I did notice this just from the parts I did review -- the audio gets pretty staticky at points.
>
> I think I might have wrongly said earlier that there's no audio error detection in DV. There is, and what I should have said was that there's no error *masking* comparable to what there is with the video. DV Analyzer should still report audio dropouts, according to documentation:
>
> https://mediaarea.net/DVAnalyzer/audio-errors
>
> (Notice that the example provided is an ATM tape, so we'll probably see it at some point.)
>
> There aren't any audio dropouts reported in 40000003818772. Maybe the dropouts were on an original tape this was copied from. Or maybe DV audio dropouts simply aren't being documented for some reason. I've spent some time today looking back through some older DV Analyzer reports, and for files we've received since the start of February, I only found one report with audio errors noted (searching on "dseq," which should always be part of an audio error entry):
>
> Tests 20200217 > 40000003816289_Panasonic > MDPI_40000003816289_002_pres.xml
>
> This was for a file created using a Panasonic deck rather than a Sony. We didn't investigate it because it was one of those Memnon already knew there was a problem with (the DV file plays at 2x speed). But I wonder if the Sony decks just don't report audio errors for some reason. It would seem a little strange otherwise that we're not seeing any in these tests.

We never did resolve this question of why our DV files never contain metadata reporting audio errors, although I brought it up during a Zoom meeting with Michel Merten et al. on October 20 when we were going over other details for the IASA / FIAT-IFTA presentation.

Over the course of April, the IUMDS team went over the first "real" DV submission packages from Memnon with a fine-toothed comb. The last unresolved points I listed in an email to Mike Casey on April 22 were:

> 1. The SDI (.mkv) file often contains one initial frame that isn't in the corresponding FireWire (.dv) file. In other words, the FireWire file starts one frame later than the SDI file. This always seems to be just a single frame, and we're not really missing any content -- it's just that one frame will only exist in the .mkv file (the PresInt rather than the Pres).
>
> 2. The SDI file often contains just one frame immediately after an end-of-recording marker, while the FireWire file contains a dozen or so identical repetitions of that same frame. This arguably isn't really

part of the recording and is just an artifact of how the two approaches to reformatting deal with the start of the "lead-out" (I'm sure that's not the right term, but you know what I mean).

3. The metadata in the .dv as read by DV Analyzer is supposed to include entries for any audio errors that have occurred -- i.e., cases where the error detection circuitry detected a mismatch between the audio data and the parity bits. None of the recently delivered files has displayed a single audio error. Maybe we've just been lucky so far, but this seems a little suspicious, and there's a chance that audio errors aren't being reported as they're supposed to be.

4. In all experiments to date, the FireWire transfer has contained only content recorded at one speed (SP or LP), whichever comes first. It routinely drops any content recorded at a different speed than the starting speed, even if this appears in the SDI transfer. We don't know if any DV tapes slated for reformatting will actually contain speed changes, but if they do, we should expect them to cause similar problems. Such tapes would probably be caught by Memnon's comparative duration check and failed.

I think that's it, as long as we're satisfied with the way anamorphic content is being handled (I don't think there are any mysteries there).

My script continued in use after this point, but we were no longer using it to assess the reformatting approach itself; instead, it became a tool for routine quality control. Nevertheless, there were a few more developments to come.

In early May 2020, I began noticing multiple missing frames at the beginnings of DV files on a regular basis—usually just one, but occasionally up to five. Memnon determined this would not be easy to fix, and Mike Casey concluded it wasn't mission-critical: "I'm comfortable with the understanding that the two streams kick in at a very slightly different time" (email of May 20). I also encountered the first tape I'd run into with content right up until its physical end, which caused my script to run into problems—I hadn't designed the script to accommodate this (as part of its routine for handling multiple DV segments, even though we didn't need that functionality any more). I tweaked the script to accommodate this situation.

The script continued to take a long time to run, and we tried a few things to increase productivity. First, I reduced the number of frames the script seeks ahead and backwards to look for a match, since in our recent experience we weren't encountering much in the way of time discrepancies. In Mid-July, I shared the script with Adam Nickel and David Adamson so that they could run it as part of their own DV quality control work and so increase the number of tapes we could analyze with it. During this period, Adam would stage sets of DV files and put them on the Y drive, from which we would then need to download them for local analysis. Moving the files around took up a good bit of time.

Meanwhile, Dennis Cromwell proposed we try to run the script on a server to see if that would go faster. In late July, Brian Wheeler ran an analysis of one sample tape on a server that took 15 minutes 19 seconds where my laptop had taken about 47 minutes. Encouraged by this, and with important guidance and input from Brian, I worked out a new version of the script that would run on the server from a command line. We agreed that it made sense to retain the functionality of being able to compare multiple DV files with a single SDI file, just in case there's any future call to do that. As a result,

the command-line takes the SDI (MKV) filename as its argument (and will then look for additional DV segments beyond the first, although in our case it will never find any). On September 9, I wrote:

> My script took 2 hours 8 minutes to analyze the latest file set on the server, as compared to 4 hours 59 minutes for the same file set on the IC desktop computer. That comes out to 42.8% as long (128 minutes divided by 299 minutes). Not quite as dramatic an improvement as Brian's initial test results suggested we might see, but still definitely a significant improvement that should make a substantial difference in how much content we can check this way.

By mid-September, I had a new workflow in place. I begin by opening four instances of Putty, logging into qc-02.mdpi.iu.edu, navigating to the workspace (cd workspace), and staging four file sets using the command **object_retrieve 40000003636695 40000003573377 45000000322153 45000000322799** (with the appropriate barcode numbers substituted, drawn from a spreadsheet furnished by Adam). I then use each Putty window to navigate to the directory for one file set (part of the folder name can't be inferred from the barcode and needs to be copied from the directory), e.g.,

cd MDPI_45000000322278.20200807-080705
cd data
dv_sdi_cmp.sh MDPI_45000000322278_01_presInt.mkv

The results will then be written to the same "data" subfolder where the DV, SDI, and DVAnalyzer XML files live. I download them to my desktop computer at the Innovation Center (via Remote Desktop) and review them. Whenever I spot an error that doesn't fit into one of the "expected" categories, I flag it in the DV quality control spreadsheet and report it to Rob, who requests the tape in question back from Memnon for further investigation. I believe a majority of tapes I've flagged have been sent for another reformatting attempt.

Some other "new" situations have turned up over time:

- June: first discovery of a dropped DV frame since start of production (these remained rare)
- July: a pattern in which a single SDI frame contains an error not present in the corresponding DV frame – oddly, the reverse situation doesn't seem to arise (this came up as a question from Dave Rice after our IASA / FIAT-IFTA presentation).
- August: cases in which the SDI file contains blank space between different recordings on a tape, while the segments are concatenated immediately next to each other in the DV file. Because I'd revised the script to reduce the number of frames it looks forwards and backwards for a match, cases in which a tape has several recordings with blank space between will usually cause it just to stop finding any matches at all, and to export a very large number of "non-matching" frames which then need to be deleted. One way to handle this type of situation without taking an inordinate amount of time, I think, would be to begin by trying to align the *audio* tracks (which would go much faster), and then to use those alignments as a starting-point for matching video segments. But that would require some further research and development, and this late in the project I've just concluded that the script will work only for analyzing the *first* recording on a tape, and not any of the later ones.

APPENDIX A:
TEXT PRESENTED AT ONLINE IASA – FIAT/IFTA CONFERENCE (OCT. 27, 2020)[2]

When assessing DV files, we wanted to distinguish between problems attributable to known shortcomings of the format, on the one hand; and unexpected problems that might point to correctable shortcomings in the transfer process itself, on the other. We knew video data would sometimes be corrupted and undergo error concealment in playback; and that more severe problems could make it impossible to capture a frame over FireWire at all, in which case we'd expect a segment break, just as we'd get if there were a gap on the tape between separate recordings: the current DV segment would end, and a new one would start with the resumption of valid signal.

Our first FireWire transfers—made before we'd instituted simultaneous SDI transfers--broke content into as many as seventy-four "segments," seemingly at random -- sometimes extremely short, sometimes missing audio, and sometimes unplayable in some or all of the software we tried. So we wanted to know: should we be concerned about this, or was it just par for the course?

We could use DVAnalyzer to examine the DV metadata and see what anomalies it reported, such as nonconsecutive time code; but as long as all we had was the DV file, we weren't in a position to judge how accurately that file reflected what the deck had read off the tape—that is, whether a problem had originated in the reading or in the writing.

But the situation changed once we had both SDI and DV files in hand. SDI files provide one continuous file per tape with a time base representing the actual chronology of playback, giving us a second source of evidence about what the deck had actually managed to read. With this in mind, I set out to write a script to compare DV and SDI files frame by frame, report any discrepancies between them, and associate them with the metadata imported from a DVAnalyzer xml report. I thought this kind of analysis might give us the best of both worlds – nice "clean," non-glitchy SDI video with the DV metadata synchronized with it for easy reference. But more immediately, I hoped it would provide insights into the digitization process that would help us assess how things were going.

For example, we expected that in cases of more severe corruption, there would be a break between DV segments, while the SDI file would instead contain whatever garbled frames the deck sent out while wrestling through the rough patch. Some frames in the SDI file might therefore correspond to breaks between DV file segments, and one question we had was how much content of this kind was missing from DV files, if any.

I ended up writing the script in Python, using the OpenCV library to read in the DV and SDI frames and the structural similarity index metric from the Scikit-Image library to compare them—through trial and error, I settled on treating a similarity score of 96% or higher as a match.

The algorithm starts by trying to find an initial matching frame between the DV and SDI files. It then checks each subsequent pair of frames to verify whether they also match. If it encounters a pair of non-matching frames, it searches ahead in the SDI file for a successful match. If it still doesn't find a match within a designated number of SDI frames, it searches backwards in the SDI file instead. The further

[2] This was my piece of the session "Defeating DV; or, How to Make Lemonade Out of Lemons," co-presented with Brecht Declercq and Mike Casey.

ahead and backwards the script is set to search, the longer the gaps it will be able to detect, but the more time-consuming processing becomes—and a single tape can easily take hours to analyze.

When we ran this script on some of the first sets of DV and SDI files we received, we were surprised to find frames in the SDI file that were missing from positions midway through DV segments. In one case, a single DV segment with no video error concealment such as we'd expect from a glitchy tape, turned out to be missing 563 frames that were present in the SDI file, including one group of 352 missing frames in a row—almost twelve seconds. Each one of these discrepancies matched a jump in the DV timecode, as reported by DVAnalyzer.

That was an unusually extreme case, but we found frames "dropped" from the middle of DV segments in other files we analyzed too—usually just a single frame, but occasionally two or more in a row. We could often rule out the possibility that spurious frames were being inserted into the SDI file by exporting the frames in question along with the frames to either side and confirming that the middle frame looked like a "real" frame, for example showing a hand in three distinct phases of a continuous movement. To us this suggested a buffering problem where the capture software hadn't kept pace with the DV stream and had ended up "dropping" one or more frames from the DV file while simultaneously continuing to capture the SDI output without interruption. A review of online message boards revealed that DV FireWire transfers were notorious for dropping frames due to buffering problems, with a software tool known as WinDV often being recommended as a solution.

We reported all these findings to Memnon, and after some further R&D they managed to solve the chronic problems we'd been seeing—as our script confirmed: we were now getting DV files with NO dropped frames. Since then, we've continued to run a version of the same script as part of our routine quality control for this format, but we're now looking mostly for other things.

One consequence of Memnon's solution was that we began receiving only one DV segment per tape, with all content concatenated together, rather than multiple segments as before. If there are unrecorded gaps between recordings on the tape, there can be large timing discrepancies between the DV and SDI files. Our script could track these situations, but only at the expense of having it seek a match across a greater number of frames, which would get time-consuming – so our use of the script has focused instead on analyzing tapes without long breaks.

We've come to ignore certain common anomalies, such as a single missing frame at the beginning of DV files or duplicate DV frames immediately after an end-of-recording flag. But when any frames fail to match under other circumstances, we follow up, and the script helps us do this by exporting pairs of non-matching frames for review.

First, it exports the DV and SDI frames side by side. Here *[referring to onscreen example]* we can see that the SDI frame has streaks across the top that are absent from the DV frame. We ended up having this tape re-digitized.

But the script also exports an animated GIF that alternates rapidly between the two frames, exposing errors that might otherwise be hard to spot. Here *[referring to another onscreen example]* we can see that one frame has errors that aren't present in the other, and a review of the side-by-side export shows that the SDI frame is once again the culprit. We had this tape re-digitized too. In this way, frame-by-frame comparison has enabled us to catch and remedy problems other tools would likely have missed.

With that, I'd like to turn the mic back over to Mike for some closing remarks.

APPENDIX B:
VERSION OF SCRIPT FOR USE ON DESKTOP OR LAPTOP COMPUTER

```
# -*- coding: utf-8 -*-
# DV-SDI Comparison Script
# by Patrick Feaster, Media Preservation Specialist, Media Digitization and Preservation Initiative, Indiana University
# mdpi.iu.edu -- pfeaster@indiana.edu
# Updated version for July 24, 2020
# This script will check simultaneous DV (FireWire) and SDI captures of a DV tape for discrepancies.
# It has been run successfully using Spyder in Anaconda Navigator with OpenCV installed.
# In addition to a report (saved as a txt file), it will also export side-by-side images of frame groups (DV on left, SDI on right):
# (1) The first matching frame pair for each DV segment.
# (2) Any pairs of frames that are expected to match but don't.
# (3) In case of #2, any matched pair that differs from the expected one .
# (4) Any case in which a DV frame doesn't match the expected SDI frame and the SDI frame duplicates the previous SDI frame.
# In case of #4, three images are provided side-by-side: the DV frame, the expected match, and the previous SDI frame it duplicates.
# === CUSTOMIZABLE VARIABLES ===
simthreshold=0.96 #threshold for matching frames between files; set lower (e.g. 0.9) for lossy files, at risk of false positives
simthreshold2=0.96 #threshold used for identifying duplicate frames in the same file; could set higher if desired
dropframethresholdfwd=10 #number of frames to be searched ahead in SDI file for a DV frame match; may need to increase in the event of very large dropped segments
dropframethresholdbkwd=10 #number of frames to be searched backwards in SDI file if no forward matches are found
framelimit=10 #number of frames checked in SDI file to find an initial match (added to number of frames advanced to find first non-repeating frame)
#Import libraries
import cv2
import numpy as np
import xml.etree.ElementTree as ET
from skimage import metrics
import tkinter
from tkinter.filedialog import askopenfilenames, askopenfilename
from datetime import datetime
from math import floor
from PIL import Image
root=tkinter.Tk()
root.withdraw()
#The user will be prompted to make three selections by navigating to them:
#(1) An SDI transfer, ideally uncompressed or losslessly compressed
sdifilename=askopenfilename(title="Select SDI transfer (*.mkv)",filetypes=(("mkv files","*.mkv"),))
#(2) One or more DV files, which should be named/selected in sequential order
dvfilename=askopenfilenames(title="Select FireWire transfer(s) (*.dv)",filetypes=(("dv files","*.dv"),))
#(3) DVAnalyzer xml reports, one per DV file, in the same order
xmlfilename=askopenfilenames(title="Select DVAnalyzer XML reports (*.xml)",filetypes=(("xml files","*.xml"),))
#Set starting values for tracking variables
currentsdiframe=0 #this keeps track of current position in SDI file
```

```
nomatch=0 #set temporarily to 1 to flag a non-matching DV file set
nothingyet=1 #has value 1 until some pair of frames has been successfully matched
video2=cv2.VideoCapture(sdifilename) #open SDI file
#get barcode number
xmlfilenamesplit=str(xmlfilename).split('MDPI_4')
xmlfilenamesplit2=xmlfilenamesplit[1].split('.xml')
barcodestring='MDPI_4'+xmlfilenamesplit2[0]
f = open(barcodestring+'.txt','a')
current_time = str(datetime.now())
f.write("=======================================\n")
f.write("ANALYSIS OF SDI = "+sdifilename+"\n")
f.write("Similarity threshold = "+str(simthreshold)+"\n")
f.write("Results run beginning "+current_time+"\n")
print('Starting analysis.')
def framereshape(frame2):
   #Function for reshaping SDI frame to dimensions of DV frame
   #The specific frame heights and adjustments reflect different file types Memnon provides to IU.
   #In other situations, it may be necessary to compare pairs of DV and SDI frames to determine how to crop the
SDI frames so that the frames match.
   #Substitute the height of the SDI frame for "486" and the row numbers corresponding to the first and last rows
in the DV frame for "4" and "484."
   if(frame2.shape[0]==486):
      frame2=frame2[4:484,:,:]
   if(frame2.shape[0]==512):
      frame2=frame2[29:509,:,:]
   return frame2
def dventries(first,last):
   #Adds all DVAnalyzer entries from "first" to "last" at current position in output report
   for x in range(first,last+1):
      if (str(x) in evframenum and evtext[int(evframenum.index(str(x)))]=="non-consecutive DV timecode") or
nomatch==1:
         framestring="DV "+str(dvfilenum+1)+":"+str(x)+frames2time(x)+" > "
      else:
         framestring="DV "+str(dvfilenum+1)+":"+str(x)+frames2time(x)+" = SDI
"+str(x+sdioffset)+frames2time(x+sdioffset)+": "
      if str(x) in framenum:
         if(frametype[int(framenum.index(str(x)))])=='first':
            f.write(framestring+'First frame recorded at '+rectime[int(framenum.index(str(x)))]+'; Original Timecode =
'+timecode[int(framenum.index(str(x)))]+"\n")
      if str(x) in flagframenum:
         if(flag[int(flagframenum.index(str(x)))]=='REC_START'):
            f.write(framestring+'Recording started at '+rectime[int(framenum.index(str(x)))]+"\n")
         if(flag[int(flagframenum.index(str(x)))]=='REC_END'):
            f.write(framestring+'Recording ended at '+rectime[int(framenum.index(str(x)))]+"\n")
      evframestr=""
      #The following is a very inelegant way to identify multiple occurrences of a value in evframenum
      while str(x) in evframenum:
         if str(x) in evframenum:
            try:
               evframestr=evframestr+evtype2[int(evframenum.index(str(x)))]+':
'+evtext[int(evframenum.index(str(x)))]+'/'
            except:
               evframestr=evframestr+evtext[int(evframenum.index(str(x)))]+'/'
```

```
            evframenum[evframenum.index(str(x))]='deleted'
        while 'deleted' in evframenum:
            if 'deleted' in evframenum:
                evframenum[evframenum.index('deleted')]=str(x)
        if str(x) in evframenum:
            f.write(framestring+evframestr+"\n")
def frames2time(frames):
    #Converts numbers of frames to hh:mm:ss format for reporting
    rawseconds=frames/29.97
    rawminutes=rawseconds/60
    hours=floor(rawminutes/60)
    minutes=floor(rawminutes-(hours*60))
    seconds=round(rawseconds-((minutes*60)+(hours*60*60)))
    outstring=" ("+str(hours)+":"+str(minutes).zfill(2)+":"+str(seconds).zfill(2)+")"
    return outstring
def animagif(frame1,frame2,title):
    #Convert frames from OpenCV to Pillow format
    frame1=frame1[:,:,::-1]
    frame2=frame2[:,:,::-1]
    frame1a=Image.fromarray(frame1)
    frame2a=Image.fromarray(frame2)
    frame1a.save(title+'.gif',save_all=True, append_images=[frame2a], duration=100, loop=0)

for dvfilenum in range(np.size(dvfilename)):
    print('Analyzing DV file '+str(dvfilenum+1)+'.')
    #this loop cycles through all the DV files if there are more than one
    video1=cv2.VideoCapture(dvfilename[dvfilenum])
    f.write("-------------------------------------------\n")
    f.write("DV SEGMENT "+str(dvfilenum+1)+"\n")
    f.write("DV = "+dvfilename[dvfilenum]+"\n")
    video_length = int(video1.get(cv2.CAP_PROP_FRAME_COUNT))
    f.write("Number of frames in DV file: "+str(video_length)+"\n")
    #import XML
    tree = ET.parse(xmlfilename[dvfilenum])
    root = tree.getroot()
    # Reset lists
    warning=[] #warnings at file level
    # Frame-specific variables
    frametype=[] #first, error, last
    framenum=[] #number of frame within file
    abtime=[] #absolute time within file
    timecode=[] #timecode value from DV file
    rectime=[] #recording time and date from DV file
    arbbit=[] #arbitrary bit (in repeating sequence)
    # Event-specific variables
    evframenum=[] #frame number corresponding to event
    evtag=[] #will always be "event"
    evid=[] #event ID
    evtype1=[] #type of event (e.g. "error")
    evtype2=[] #subtype of event (e.g. "video error concealment")
    evtext=[] #text describing event
    # Flag-specific variables
    flagframenum=[] #frame number corresponding to flag
```

```
flagtag=[] #will always be 'flag'
flag=[] #flag text
#Read in the XML
for elem in root.iter(tag='file'):
   for elem2 in elem.iter(tag='warnings'):
      for elem3 in elem2:
         warning.append(elem3.text)
   for elem2 in elem.iter(tag='frames'):
      for elem3 in elem2.iter(tag='frame'):
         if elem3.get('type') is not None:
            frametype.append(elem3.get('type'))
         else:
            frametype.append('none')
         for elem4 in elem3:
            if(elem4.tag=='frame'):
               framenum.append(elem4.text)
            elif(elem4.tag=='abs_time'):
               abtime.append(elem4.text)
            elif(elem4.tag=='dv_timecode'):
               timecode.append(elem4.text)
            elif(elem4.tag=='recdate_rectime'):
               rectime.append(elem4.text)
            elif(elem4.tag=='arbitrary_bit'):
               arbbit.append(elem4.text)
            elif(elem4.tag=='events'):
               for elem5 in elem4:
                  evframenum.append(framenum[-1])
                  evtag.append(elem5.tag)
                  evtext.append(elem5.text)
                  evtype1.append(elem5.get('type'))
                  evid.append(elem5.get('event_id'))
                  evtype2.append(elem5.get('event_type'))
            elif(elem4.tag=='flags'):
               for elem5 in elem4.iter(tag='flag'):
                  flagframenum.append(framenum[-1])
                  flagtag.append(elem5.tag)
                  flag.append(elem5.text)
#Report any warning associated with current DV segment
if(len(warning)>0):
   for z in range(len(warning)):
      f.write("DV "+str(dvfilenum+1)+" > WARNING: "+warning[z]+"\n")
#If this is the first DV segment, or a later one when no match has been found
print('Seeking initial match.')
if(nothingyet==1):
   #Find first non-repeating frame in DV file (in case of color bars, etc.)
   ret,frame1=video1.read()
   if(ret==False):
      f.write("Failed to read any DV frame.\n")
      break
   ret,frame2=video1.read()
   currentFrame=2
   if(ret):
      #As long as both frames are equal
```

```
        while(np.array_equal(frame1,frame2))and(ret==True):
            #Advance a frame and check again
            if(currentFrame>framelimit):
                break
            ret,frametemp=video1.read()
            if ret:
                frame2=frametemp
                currentFrame+=1
    else: #If there was no frame 2 to read, select frame 1
        frame2=frame1
    #Set back to first frame if the first tested pair was already unequal or if there was only one frame
    if(currentFrame==2):
        currentFrame=1
        frame2=frame1
    #Find best match for selected frame in SDI file, starting either at its beginning (for first DV segment) or
wherever it currently is
    #Note that this script does not search BACKWARDS in the SDI file
    simcheck=0
    itercount=max(-1,currentsdiframe)
    frameschecked=0
    while(simcheck<simthreshold):
        ret,frame3=video2.read()
        if (ret)&(frameschecked<framelimit+currentFrame):
            frame3=framereshape(frame3)
            #Check for similarity
            simcheck=metrics.structural_similarity(frame2,frame3,multichannel=True)
            itercount+=1
            frameschecked+=1
        else:
            itercount-=1
            break
    #Define pair of corresponding frames
    #If no best match was found, just set both files to 0
    if simcheck<simthreshold:
        f.write("Failed to find an initial match with SDI file after checking "+str(framelimit)+" frames.\n")
        refFrame1=0
        refFrame2=0
    #Otherwise, set file starting points based on detected match
    else:
        refFrame1=currentFrame
        refFrame2=itercount
    print('Comparing frames.  This may take a while.')
    #Reset video reader for both files with appropriate offset
    #Variable nowdvframe is set to minus one because first comparison loop below will iterate it by one (to "0")
with a successful match
    sdioffset=refFrame2-refFrame1
    if (refFrame1<refFrame2):
        video1.set(cv2.CAP_PROP_POS_FRAMES,0)
        video2.set(cv2.CAP_PROP_POS_FRAMES,refFrame2-refFrame1)
        nowdvframe=-1
    elif (refFrame1>refFrame2):
        video1.set(cv2.CAP_PROP_POS_FRAMES,refFrame1-refFrame2)
        video2.set(cv2.CAP_PROP_POS_FRAMES,0)
```

```
            nowdvframe=(refFrame1-refFrame2)-1
            dventries(0,nowdvframe)
         else:
            video1.set(cv2.CAP_PROP_POS_FRAMES,0)
            video2.set(cv2.CAP_PROP_POS_FRAMES,0)
            nowdvframe=-1
      else:#if this is a subsequent DV segment
         #add past DV frame number to the sdioffset and ADD ONE
         sdioffset=sdioffset+nowdvframe+1
         #reset the DV frame number to minus one
         nowdvframe=-1
      #Flag current loop as the first for this DV file.
      firstpair=1
      while(True):
         #Read a pair of frames, one from each video
         oldsdi=frame2 #save to check for repeated SDI frame
         oldsdioffset=sdioffset #save for subsequent restoration if no matches found
         ret1,frame1=video1.read()
         #If there's another frame in the current DV segment
         if ret1==True:
            nowdvframe +=1
            #Read another SDI frame
            ret2,frame2=video2.read()
            if(ret2):
               frame2=framereshape(frame2)
               #Compare the two frames
               simcheck=metrics.structural_similarity(frame1,frame2,multichannel=True)
               droppedFrames=0
               expectedMatch=nowdvframe+sdioffset
               oldoffset=sdioffset
               firstnonmatch=0
               if(simcheck<simthreshold) and (firstpair==0):
                  #If the frames don't match
                  #Check to see if SDI frame is duplicate of previous SDI frame
                  simcheckalt=metrics.structural_similarity(frame2,oldsdi,multichannel=True)
                  dupesfound=0
                  #If it's a duplicate
                  while(simcheckalt>=simthreshold2):
                     #Report the fact
                     f.write("SDI frame "+str(nowdvframe+sdioffset)+" is a duplicate of SDI frame
"+str(nowdvframe+sdioffset-1)+"\n")
                     #Add one to the offset count
                     sdioffset +=1
                     #And read another frame forward, checking IT for duplication
                     ret1,frame2=video2.read()
                     frame2=framereshape(frame2)
                     simcheckalt=metrics.structural_similarity(frame2,oldsdi,multichannel=True)
                     dupesfound=1
                  #Recalculate simcheck if necessitated by the foregoing
                  if(dupesfound==1):
                     simcheck=metrics.structural_similarity(frame1,frame2,multichannel=True)
               if(simcheck>simthreshold) and (firstpair==1):
                  #if a subsequent match turns up, report it
```

```
            f.write("DV "+str(dvfilenum+1)+":"+str(nowdvframe)+frames2time(nowdvframe)+" = SDI
"+str(nowdvframe+sdioffset)+frames2time(nowdvframe+sdioffset)+"\n")
          while(simcheck<simthreshold):
            #Condition of while loop indicates that the frames don't match, and at this point we know the SDI
frame isn't a duplicate of the previous SDI frame
            #If this is the first failure to match the DV frame to an SDI frame, and not the first DV frame in the file,
report
            if(firstnonmatch==0) and (firstpair==0):
              outim=np.concatenate((frame1,frame2), axis=1)

cv2.imwrite(barcodestring+'_'+str(dvfilenum+1)+"~"+str(nowdvframe)+"_"+str(nowdvframe+sdioffset)+"_no_matc
h.jpg",outim)

animagif(frame1,frame2,barcodestring+'_'+str(dvfilenum+1)+"~"+str(nowdvframe)+"_"+str(nowdvframe+sdioffset
)+"_no_match")
              firstnonmatch=1
            #Increment dropped frames
            droppedFrames+=1
            #Read another frame from the SDI file and adjust its size
            ret1,frame2=video2.read()
            #If something has been read
            if(ret1):
              frame2=framereshape(frame2)
              simcheck=metrics.structural_similarity(frame1,frame2,multichannel=True)
              #Increase the offset by one to reflect the newly read frame
              sdioffset +=1
              #If it's a match, report
              if(simcheck>=simthreshold):
                if(firstpair==0):
                  #If it's not the first match in the DV file
                  f.write("DV "+str(dvfilenum+1)+":"+str(nowdvframe)+frames2time(nowdvframe)+" > expected
SDI = "+str(expectedMatch)+frames2time(expectedMatch)+"; actual SDI =
"+str(nowdvframe+sdioffset)+frames2time(nowdvframe+sdioffset)+"; discrepant
frames="+str(droppedFrames)+"\n")
                else:
                  #If it's the first match in the DV file
                  f.write("DV "+str(dvfilenum+1)+":"+str(nowdvframe)+frames2time(nowdvframe)+" = SDI
"+str(nowdvframe+sdioffset)+frames2time(nowdvframe+sdioffset)+"\n")
              #If it's not a match and the limit of searches has been reached
              if(droppedFrames>dropframethresholdfwd):
                #Report
                f.write("DV "+str(dvfilenum+1)+":"+str(nowdvframe)+frames2time(nowdvframe)+" > expected SDI
= "+str(expectedMatch)+frames2time(expectedMatch)+"; SDI match not detected within
"+str(dropframethresholdfwd)+" frames\n")
                #Now try searching backwards in the SDI file for a match
                backwardsfound=0
                for backstep in range(1,dropframethresholdbkwd+1):
                  video2.set(cv2.CAP_PROP_POS_FRAMES,nowdvframe+oldoffset-backstep)
                  ret1,frame2=video2.read()
                  frame2=framereshape(frame2)
                  simcheck=metrics.structural_similarity(frame1,frame2,multichannel=True)
                  #Report a backwards match, if any
                  if(simcheck>=simthreshold):
```

```
                        sdioffset=oldoffset-backstep
                        f.write("DV "+str(dvfilenum+1)+":"+str(nowdvframe)+frames2time(nowdvframe)+" > actual
SDI = "+str(nowdvframe+sdioffset)+frames2time(nowdvframe+sdioffset)+"; backwards match\n")
                        backwardsfound=1
                        break
                    #if there's no backwards match either
                    if(backwardsfound==0):
                       sdioffset=oldoffset
                    #Reset offset to continue processing
                    video2.set(cv2.CAP_PROP_POS_FRAMES,nowdvframe+sdioffset+1)
                    break
                else:
                   f.write("SDI "+str(nowdvframe+sdioffset)+" ~ Not read.\n")
                   sdioffset +=1
                   break
          #Since an analysis loop has completed, stipulate that this is no longer the first match for the DV segment (or
overall)
          firstpair=0
          nothingyet=0
          #Report any errors, etc., from DVAnalyzer report for latest frame analyzed
          dventries(nowdvframe,nowdvframe)
       else:
          #If we've reached the end of a DV segment, report what its last frame was
          if(nowdvframe>=0):
             #If any DV frames were found to match any SDI frames
             f.write("DV "+str(dvfilenum+1)+":"+str(nowdvframe)+frames2time(nowdvframe)+" = SDI
"+str(nowdvframe+sdioffset)+frames2time(nowdvframe+sdioffset)+": Last frame recorded at "+rectime[-1]+";
Original Timecode = "+timecode[-1]+"\n")
          else:
             #If there was no match, report and revert variables
             sdioffset=oldsdioffset
             nomatch=1
             dventries(0,0)
             nomatch=0
             f.write("DV "+str(dvfilenum+1)+":0 ~ No SDI match found.\n")
             nowdvframe=0
          break
#After analyzing the last DV segment, count remaining frames in SDI file and compare the total number of read
frames to the expected frame total
continuer=1
sdiframenum=nowdvframe+sdioffset
while(continuer==1):
   ret1,frame1=video2.read()
   if(ret1==True):
      sdiframenum +=1
   else:
      continuer=0
video_length = int(video2.get(cv2.CAP_PROP_FRAME_COUNT))
f.write("Final SDI frame = "+str(sdiframenum)+frames2time(sdiframenum)+"\n")
f.write("Number of frames in SDI file: "+str(video_length)+"\n")
current_time = str(datetime.now())
print('Analysis completed.')
f.write("Analysis completed "+current_time+"\n")
```

```
#Release the two video readers
video1.release()
video2.release()
#Close the report
f.close()
```

APPENDIX C:
VERSION OF SCRIPT FOR USE ON SERVER (REVISED OCTOBER 2020)

```
#Import libraries
import cv2
import argparse
import numpy as np
import xml.etree.ElementTree as ET
from skimage import metrics
from datetime import datetime
from math import floor
from PIL import Image
from pathlib import Path

def main(sdifilename,simthreshold,framelimit):
   f = Path(sdifilename)
   stem = f.name[:20]  # the first 20 characters of the filename part
   part = 1
   xmlfiles = []
   dvfiles = []
   while True:
      # generate filenames for this part.
      xmlfile = f.parent / f"{stem}{part:02d}_pres.xml"
      dvfile = f.parent / f"{stem}{part:02d}_pres.dv"
      # since dvfile and xmlfile are Path objects, we can use exists()
      # to see of those filenames actually exist on the system.
      if not xmlfile.exists() or not dvfile.exists():
         # one or both of the next part files doesn't exist...so exit.
         break
      # I'm converting the Path objects back to strings, but if
      # you'd want them as path objects (which can be used in
      # open or other file operations), you'd just drop the str()
      xmlfiles.append(str(xmlfile))
      dvfiles.append(str(dvfile))
      part += 1
   if len(xmlfiles) == 0 or len(dvfiles) == 0:
      print("No xml or dv file pairs found")
      exit(1)
   currentsdiframe=0 #this keeps track of current position in SDI file
   nothingyet=1 #has value 1 until some pair of frames has been successfully matched
   nothingyetthisdv=1
   video2=cv2.VideoCapture(sdifilename) #open SDI file
   f = open(f"{stem}report.txt",'a')
   current_time = str(datetime.now())
   f.write("=========================================\n")
   f.write(f"ANALYSIS OF SDI = {sdifilename}\n")
```

```
f.write(f"Similarity threshold = {simthreshold}\n")
f.write(f"Results run beginning {current_time}\n")
print('Starting analysis.')
for dvfilenum in range(np.size(dvfiles)):
   print(f'Analyzing DV file {dvfilenum+1}.')
   #this loop cycles through all the DV files if there are more than one
   video1=cv2.VideoCapture(dvfiles[dvfilenum])
   f.write("-------------------------------------------\n")
   f.write(f"DV SEGMENT {dvfilenum+1}\n")
   f.write(f"DV = {dvfiles[dvfilenum]}\n")
   video_length = int(video1.get(cv2.CAP_PROP_FRAME_COUNT))
   f.write(f"Number of frames in DV file: {video_length}\n")
   #import XML
   tree = ET.parse(xmlfiles[dvfilenum])
   root = tree.getroot()
   frames = {}
   for framenode in root.findall('file/frames/frame'):
      frame = {'type': framenode.get('type'),
            'framenum': int(framenode.findtext('frame')),
            'abs_time': framenode.findtext('abs_time'),
            'dv_timecode': framenode.findtext('db_timecode'),
            'recdate_rectime': framenode.findtext('recdata_rectime'),
            'arbitrary_bit': framenode.findtext('arbitrary_bit'),
            'events': []
      }
      for eventnode in framenode.findall('events/event'):
         event = {
            'type': eventnode.get('type'),
            'event_id': eventnode.get('event_id'),
            'event_type': eventnode.get('event_type'),
            'text': eventnode.findtext('.')
         }
         frame['events'].append(event)
      for flagnode in framenode.findall('flags/flag'):
         frame['events'].append(flagnode.findtext('.'))
      frames[frame['framenum']] = frame
   #If this is the first DV segment, or a later one when no match has been found
   print('Seeking initial match.')
   if(nothingyet==1):
      #Find first non-repeating frame in DV file (in case of color bars, etc.)
      ret,frame1=video1.read()
      if(ret==False):
         f.write("Failed to read any DV frame.\n")
         break
      ret,frame2=video1.read()
      currentFrame=2
      if(ret):
         #As long as both frames are equal
         while(np.array_equal(frame1,frame2))and(ret==True):
            #Advance a frame and check again
            if(currentFrame>framelimit):
               currentFrame=1
               break
```

```
            ret,frametemp=video1.read()
            if ret:
              frame2=frametemp
              currentFrame+=1
        else: #If there was no frame 2 to read, select frame 1
          frame2=frame1
        #Set back to first frame if the first tested pair was already unequal or if there was only one frame
        if(currentFrame==2):
          currentFrame=1
          frame2=frame1
        #Find best match for selected frame in SDI file, starting either at its beginning (for first DV segment) or
wherever it currently is
        #Note that this script does not search BACKWARDS in the SDI file
        simcheck=0
        itercount=max(-1,currentsdiframe)
        frameschecked=0
        while(simcheck<simthreshold):
          ret,frame3=video2.read()
          if(ret)&(frameschecked<framelimit+currentFrame):
            frame3=framereshape(frame3)
            #Check for similarity
            simcheck=metrics.structural_similarity(frame2,frame3,multichannel=True)
            itercount+=1
            frameschecked+=1
          else:
            itercount-=1
            break
        #Define pair of corresponding frames
        #If no best match was found, just set both files to 0
        if simcheck<simthreshold:
          f.write("Failed to find an initial match with SDI file after checking "+str(framelimit)+" frames.\n")
          refFrame1=0
          refFrame2=0
        #Otherwise, set file starting points based on detected match
        else:
          refFrame1=currentFrame
          refFrame2=itercount
        print('Comparing frames.  This may take a while.')
        #Reset video reader for both files with appropriate offset
        #Variable nowdvframe is set to minus one because first comparison loop below will iterate it by one (to "0")
with a successful match
        sdioffset=refFrame2-refFrame1
        if (refFrame1<refFrame2):
          video1.set(cv2.CAP_PROP_POS_FRAMES,0)
          video2.set(cv2.CAP_PROP_POS_FRAMES,refFrame2-refFrame1)
          nowdvframe=-1
        elif (refFrame1>refFrame2):
          video1.set(cv2.CAP_PROP_POS_FRAMES,refFrame1-refFrame2)
          video2.set(cv2.CAP_PROP_POS_FRAMES,0)
          nowdvframe=(refFrame1-refFrame2)-1
          dventries(0,nowdvframe,frames,f,dvfilenum)
        else:
          video1.set(cv2.CAP_PROP_POS_FRAMES,0)
```

```
            video2.set(cv2.CAP_PROP_POS_FRAMES,0)
            nowdvframe=-1
        else:#if this is a subsequent DV segment
          #add past DV frame number to the sdioffset and ADD ONE
          sdioffset=sdioffset+nowdvframe+1
          #reset the DV frame number to minus one
          nowdvframe=-1
        #Flag current loop as the first for this DV file.
        firstpair=1
        while(True):
          #Read a pair of frames, one from each video
          oldsdi=frame2 #save to check for repeated SDI frame
          oldsdioffset=sdioffset #save for subsequent restoration if no matches found
          ret1,frame1=video1.read()
          #If there's another frame in the current DV segment
          if ret1==True:
            nowdvframe +=1
            #Read another SDI frame
            ret2,frame2=video2.read()
            if(ret2):
              frame2=framereshape(frame2)
              #Compare the two frames
              simcheck=metrics.structural_similarity(frame1,frame2,multichannel=True)
              droppedFrames=0
              expectedMatch=nowdvframe+sdioffset
              oldoffset=sdioffset
              firstnonmatch=0
              if(simcheck<simthreshold) and (firstpair==0):
                #If the frames don't match
                #Check to see if SDI frame is duplicate of previous SDI frame
                simcheckalt=metrics.structural_similarity(frame2,oldsdi,multichannel=True)
                dupesfound=0
                #If it's a duplicate
                while(simcheckalt>=simthreshold):
                  #Report the fact
                  f.write(f"SDI frame {nowdvframe+sdioffset} is a duplicate of SDI frame {nowdvframe+sdioffset-
1}\n")
                  #Add one to the offset count
                  sdioffset +=1
                  #And read another frame forward, checking IT for duplication
                  ret1,frame2=video2.read()
                  frame2=framereshape(frame2)
                  simcheckalt=metrics.structural_similarity(frame2,oldsdi,multichannel=True)
                  dupesfound=1
                #Recalculate simcheck if necessitated by the foregoing
                if(dupesfound==1):
                  simcheck=metrics.structural_similarity(frame1,frame2,multichannel=True)
                  f.write(f"DV {dvfilenum+1}:{nowdvframe}{frames2time(nowdvframe)} = SDI
{nowdvframe+sdioffset}{frames2time(nowdvframe+sdioffset)}\n")
              if(simcheck>simthreshold) and (firstpair==1):
                nothingyetthisdv=1
                #if a subsequent match turns up, report it
```

```
                f.write(f"DV {dvfilenum+1}:{nowdvframe}{frames2time(nowdvframe)} = SDI
{nowdvframe+sdioffset}{frames2time(nowdvframe+sdioffset)}\n")
            while(simcheck<simthreshold):
                #Condition of while loop indicates that the frames don't match, and at this point we know the SDI
frame isn't a duplicate of the previous SDI frame
                #If this is the first failure to match the DV frame to an SDI frame, and not the first DV frame in the file,
report
                if(firstnonmatch==0) and (firstpair==0):
                    outim=np.concatenate((frame1,frame2), axis=1)
                    cv2.imwrite(f"{stem}{dvfilenum+1}~{nowdvframe}_{nowdvframe+sdioffset}_no_match.jpg",outim)

animagif(frame1,frame2,f"{stem}{dvfilenum+1}~{nowdvframe}_{nowdvframe+sdioffset}_no_match")
                    firstnonmatch=1
                #Increment dropped frames
                droppedFrames+=1
                #Read another frame from the SDI file and adjust its size
                ret1,frame2=video2.read()
                #If something has been read
                if(ret1):
                    frame2=framereshape(frame2)
                    simcheck=metrics.structural_similarity(frame1,frame2,multichannel=True)
                    #Increase the offset by one to reflect the newly read frame
                    sdioffset +=1
                    #If it's a match, report
                    if(simcheck>=simthreshold):
                        nothingyetthisdv=1
                        if(firstpair==0):
                            #If it's not the first match in the DV file
                            f.write(f"DV {dvfilenum+1}:{nowdvframe}{frames2time(nowdvframe)} > expected SDI =
{expectedMatch}{frames2time(expectedMatch)}; actual SDI =
{nowdvframe+sdioffset}{frames2time(nowdvframe+sdioffset)}; discrepant frames={droppedFrames}\n")
                        else:
                            #If it's the first match in the DV file
                            f.write(f"DV {dvfilenum+1}:{nowdvframe}{frames2time(nowdvframe)} = SDI
{nowdvframe+sdioffset}{frames2time(nowdvframe+sdioffset)}\n")
                #If it's not a match and the limit of searches has been reached
                if(droppedFrames>framelimit):
                    #Report
                    f.write(f"DV {dvfilenum+1}:{nowdvframe}{frames2time(nowdvframe)} > expected SDI =
{expectedMatch}{frames2time(expectedMatch)}; SDI match not detected within {framelimit} frames\n")
                    #Now try searching backwards in the SDI file for a match
                    backwardsfound=0
                    for backstep in range(1,framelimit+1):
                        video2.set(cv2.CAP_PROP_POS_FRAMES,nowdvframe+oldoffset-backstep)
                        ret1,frame2=video2.read()
                        frame2=framereshape(frame2)
                        simcheck=metrics.structural_similarity(frame1,frame2,multichannel=True)
                        #Report a backwards match, if any
                        if(simcheck>=simthreshold):
                            nothingyetthisdv=1
                            sdioffset=oldoffset-backstep
                            f.write(f"DV {dvfilenum+1}:{nowdvframe}{frames2time(nowdvframe)} > actual SDI =
{nowdvframe+sdioffset}{frames2time(nowdvframe+sdioffset)}; backwards match\n")
```

```
                        backwardsfound=1
                        break
                    #if there's no backwards match either
                    if(backwardsfound==0):
                       sdioffset=oldoffset
                    #Reset offset to continue processing
                    video2.set(cv2.CAP_PROP_POS_FRAMES,nowdvframe+sdioffset+1)
                    break
                else:
                   f.write(f"SDI {nowdvframe+sdioffset} ~ Not read.\n")
                   sdioffset +=1
                   break
          #Since an analysis loop has completed, stipulate that this is no longer the first match for the DV segment
(or overall)
          firstpair=0
          nothingyet=0
          #Report any errors, etc., from DVAnalyzer report for latest frame analyzed
          dventries(nowdvframe,nowdvframe,frames,f,dvfilenum)
        else:
          #If we've reached the end of a DV segment
          if(nothingyetthisdv==0):
             #If there was no match, report and revert variables
             sdioffset=oldsdioffset
             dventries(0,0,frames,f,dvfilenum)
             f.write(f"DV {dvfilenum+1}:0 ~ No SDI match found.\n")
             nowdvframe=0
          break
   #After analyzing the last DV segment, count remaining frames in SDI file and compare the total number of read
frames to the expected frame total
   continuer=1
   sdiframenum=nowdvframe+sdioffset
   while(continuer==1):
      ret1,frame1=video2.read()
      if(ret1==True):
         sdiframenum +=1
      else:
         continuer=0
   video_length = int(video2.get(cv2.CAP_PROP_FRAME_COUNT))
   f.write(f"Final SDI frame = {sdiframenum}{frames2time(sdiframenum)}\n")
   f.write(f"Number of frames in SDI file: {video_length}\n")
   current_time = str(datetime.now())
   print('Analysis completed.')
   f.write(f"Analysis completed {current_time}\n")
   #Release the two video readers
   video1.release()
   video2.release()
   #Close the report
   f.close()

def framereshape(frame):
   #Function for reshaping SDI frame to dimensions of DV frame
   #The specific frame heights and adjustments reflect different file types Memnon provides to IU.
```

```
   #In other situations, it may be necessary to compare pairs of DV and SDI frames to determine how to crop the
SDI frames so that the frames match.
   #Substitute the height of the SDI frame for "486" and the row numbers corresponding to the first and last rows
in the DV frame for "4" and "484."
   if(frame.shape[0]==486):
      frame=frame[4:484,:,:]
   if(frame.shape[0]==512):
      frame=frame[29:509,:,:]
   return frame

def frames2time(frames):
   #Converts numbers of frames to hh:mm:ss format for reporting
   rawseconds=frames/29.97
   rawminutes=rawseconds/60
   hours=floor(rawminutes/60)
   minutes=floor(rawminutes-(hours*60))
   seconds=round(rawseconds-((minutes*60)+(hours*60*60)))
   outstring=" ("+str(hours)+":"+str(minutes).zfill(2)+":"+str(seconds).zfill(2)+")"
   return outstring

def animagif(frame1,frame2,title):
   #Convert frames from OpenCV to Pillow format
   frame1=frame1[:,:,::-1]
   frame2=frame2[:,:,::-1]
   frame1a=Image.fromarray(frame1)
   frame2a=Image.fromarray(frame2)
   #Combine into animated GIF
   frame1a.save(f'{title}.gif',save_all=True, append_images=[frame2a], duration=100, loop=0)

def dventries(first,last,frames,f,dvfilenum):
   #Adds all DVAnalyzer entries from "first" to "last" at current position in output report
   for counter in range(first,last+1):
      if counter in frames.keys():
         if(frames[counter]['type']=='first'):
            f.write(f"DV {dvfilenum+1}:{counter}{frames2time(counter)} >  FIRST FRAME\n")
         if(frames[counter]['type']=='last'):
            f.write(f"DV {dvfilenum+1}:{counter}{frames2time(counter)} >  LAST FRAME\n")
         for event in range(len(frames[counter]['events'])):
            try:
               f.write(f"DV {dvfilenum+1}:{counter}{frames2time(counter)} >
{frames[counter]['events'][event]['type']}: {frames[counter]['events'][event]['event_type']}:
{frames[counter]['events'][event]['text']}\n")
            except:
               f.write(f"DV {dvfilenum+1}:{counter}{frames2time(counter)} >  {frames[counter]['events'][event]}\n")

if __name__ == "__main__":
   parser = argparse.ArgumentParser()
   parser.add_argument("--framelimit", type=int, default=10, help="set the frame limit")
   parser.add_argument("--matchiness", type=float, default=0.96, help="how matchy to be")
   parser.add_argument('sdi', help="Filename to process")
   args = parser.parse_args()
   main(args.sdi,args.matchiness,args.framelimit)
```

APPENDIX D: REPORT ON ANALYSIS OF DV TEST FILES FROM FEBRUARY 12, 2020

For convenience, we will refer in this document to FireWire transfers and resulting files with extension .dv as "DV," and to SDI transfers and resulting files with extension .mkv as "SDI."

Expected vs. Unexpected Problems with DV

We expect that DV files will exhibit a number of problems, which is our main reason for wanting SDI files to complement them. However, we simultaneously value the DV files for the metadata they contain, as well as for their maintenance of the video information in its native format, closest to the original signal, with least alteration, consistent with preservation principles. The fact that we have an SDI file doesn't eliminate our need for an optimally captured DV file. When we review DV files, it's important for us to distinguish carefully between (1) problems that are expected and perhaps inevitable due to known shortcomings of the format and (2) problems that aren't expected and may point to correctable shortcomings in the transfer process itself.

Here's a list of some *expected* problems and outcomes:

Problem	**DV file**	**SDI file**
Corrupted video data	Pixels are interpolated, and metadata is written to document the quantity of interpolated pixels per frame (DVAnalyzer will report this, e.g., "video error concealment: 1.04% (14 A errors)")	Pixels are interpolated in the same way, but the interpolation isn't documented
Corrupted metadata that doesn't practically affect reformatting	Metadata is written containing anomalies that can be detected by DVAnalyzer, e.g., "DV timecode incoherency"	No effect
Change in technical metadata, e.g., shift from SP to LP	**New segment** is created with the new, different specification	No effect (beyond the ordinary artifacts of a start/stop point)
Corrupted technical metadata, e.g., garbled audio specification	**New segment** is created to reflect the "wrong" specification, possibly very short (e.g., a single frame)	No effect (or maybe a glitch if the player tries to adjust to the "wrong" specification)
Other, more severe data corruption	Current segment is closed, nothing is written to disc for a while, and a **new segment** begins if and when coherent data stream resumes	Frames will continue to be written but are likely to be garbled

In the past, we understand that "rules" implemented by Memnon may have led to separately-written DV segments being concatenated before delivery. Our understanding is that this is no longer being done, and that the segments we're getting now are the ones originally written to disc.

The main *expected* problem with DV files is that a given tape might result in a number of separate DV segments. In that case, the SDI file will have the advantage of maintaining a single continuous sequence. We also expect that there will be situations in which the DV stream is too severely corrupted for DV frames to be written at all, so that the SDI file may contain frames—probably distorted ones—that aren't present in any DV file. This situation should always lead to the creation of a new segment.

One thing we shouldn't expect to find is that a DV segment lacks one or more frames *midway through* that appear in the corresponding part of the SDI file. The expected response of the capture software to a "problem" frame in the DV stream is to create a segment break. However, we're routinely finding frames in the SDI file that are missing from positions *midway through* DV segments. There appear to be two possible explanations:

1. A buffer underrun in which the capture software isn't able to keep up with the DV stream and so ends up "dropping" one or more frames. If this happens, we should expect to find "dropped" frames linked to jumps in the DV timecode, which DVAnalyzer will report as "non-consecutive DV timecode," usually in conjunction with "non-consecutive arbitrary bit." We should also find a frame in the SDI file that isn't in the DV file but appears to be a "real" frame, for example by showing one phase in a continuous motion that spans the preceding and following frame—say, a person's hand in three distinct positions moving in the same direction.
2. An issue on the SDI side in which frames are being wrongly inserted. In this case, we shouldn't expect to see any timecode anomaly in the DV file. Instead, we should find a spurious frame in the SDI file—a repetition of the previous frame, a wrongly deinterlaced combination of pieces of two frames, etc.

In our first tests, carried out in January, we found many cases matching the "symptoms" of type #1. These were all the more striking because the tape appeared to be unproblematic otherwise. The first test produced a DV file with no video error concealment at all, but with the following "drops":

```
DV 21 = SDI 1
DV 23499: expected SDI = 23479; actual SDI = 23519; dropped frames=40
DV 23500: expected SDI = 23520; actual SDI = 23521; dropped frames=1
DV 23501: expected SDI = 23522; actual SDI = 23544; dropped frames=22
DV 23502: expected SDI = 23545; actual SDI = 23547; dropped frames=2
DV 23616: expected SDI = 23661; actual SDI = 23673; dropped frames=12
DV 23645: expected SDI = 23702; actual SDI = 23720; dropped frames=18
DV 23699: expected SDI = 23774; actual SDI = 23842; dropped frames=68
DV 23709: expected SDI = 23852; actual SDI = 23853; dropped frames=1
DV 23710: expected SDI = 23854; actual SDI = 23855; dropped frames=1
DV 23889: expected SDI = 24034; actual SDI = 24063; dropped frames=29
DV 24835: expected SDI = 25009; actual SDI = 25361; dropped frames=352
DV 24836: expected SDI = 25362; actual SDI = 25369; dropped frames=7
DV 27188: expected SDI = 27721; actual SDI = 27726; dropped frames=5
DV 41502: expected SDI = 42040; actual SDI = 42041; dropped frames=1
DV 41742: expected SDI = 42281; actual SDI = 42282; dropped frames=1
DV 55764: expected SDI = 56304; actual SDI = 56305; dropped frames=1
DV 69785: expected SDI = 70326; actual SDI = 70327; dropped frames=1
```

DV 77517: expected SDI = 78059; actual SDI = 78060; dropped frames=1 TOTAL DROPPED FRAMES: 563

Every one of these "drops" matched a point at which DVAnalyzer also reported a timecode anomaly. In subsequent tests with the same tape, we continued to see "drops," but fewer of them, and usually with only a single missing frame. It's possible that the longer "drops" we were discovering earlier on in the process were the result of Memnon implementing concatenation rules, although none of the rules we've had described to us would readily account for the specific patterns we were seeing.

More recently, we've begun to see additional "drops" that don't correspond to timecode anomalies reported by DVAnalyzer. We've seen these in a couple past test batches but will limit our description here to cases found in the latest three-tape test batch.

Method of Analysis

Our algorithm uses a structural similarity index to compare pairs of video frames. Matching frames typically score above 99%, while non-matching frames typically score below 90%, even if the difference is barely perceptible to the eye. We've been using an index of 96% as our threshold. The algorithm identifies the first non-duplicate frame in a DV file and matches it against a corresponding frame in the corresponding SDI file. It then checks each subsequent pair of frames to verify whether they match. If it encounters a pair of frames that are less than 96% structurally similar, it searches ahead in the SDI file for a successful match; if it doesn't find one, it then searches backwards (a hit is then reported as a "backwards match"). For present purposes, we're ignoring cases in which a single DV frame couldn't be matched as long as the next DV frame matched the expected SDI frame. We're considering only those cases where a DV frame matched a different frame in the SDI file than the expected one (usually one frame ahead), and then subsequent frames continued to match frames that were consistently "off" by the same amount. In our experience, this method usually pinpoints the exact frame where a discrepancy has occurred. In some cases where many sequential frames are very similar, it's possible that a discrepancy might not be noted until some frames *after* the point at which it has occurred, although we have yet to verify such a case (at least, while using a 96% threshold—this has definitely happened with a lower threshold). Even in such cases, the total *quantity* of non-corresponding frames should still be accurate. This method will find discrepancies between DV and SDI files, but detection of a "dropped frame" can mean either that a frame is missing from the DV file or that the SDI file contains an extra, spurious frame.

Whenever our script detects a mismatch, it exports an image containing the flagged pair of frames side-by-side with the DV frame on the left and the SDI frame on the right. It also exports another image, structured in the same way, if it finds a match for the same DV frame afterwards.

We also have a separate script we can use to export one or more specific numbered frames if we want to examine them further. It can be particularly useful to export an SDI frame that appears to be missing from a DV file, together with the preceding and following SDI frames, if we want to assess whether the "middle" frame represents a real, meaningful video frame or something interpolated or garbled.

Analysis of Test 4 (20200203)

40000003823665 (mandolin maker)

40000003474808 (choir performance)
50000000025060 (Test tape with LP/SP transitions)

40000003474808 (choir performance)

The DVAnalyzer report for this tape noted only a couple very minor issues: a change in recording date and time at frame 6140 and an error concealment at frame 86470. However, we found four frames in the SDI file that aren't in the DV file. These don't correspond to timecode anomalies in the DV file (there weren't any).

DV 1:14634 (0:08:08) > expected SDI = **14635** (0:08:08); actual SDI = 14636 (0:08:08); dropped frames=1
DV 1:36974 (0:20:34) > expected SDI = **36976** (0:20:34); actual SDI = 36977 (0:20:34); dropped frames=1
DV 1:59309 (0:32:59) > expected SDI = **59312** (0:32:59); actual SDI = 59313 (0:32:59); dropped frames=1
DV 1:81643 (0:45:24) > expected SDI = **81647** (0:45:24); actual SDI = 81648 (0:45:24); dropped frames=1

It's probably significant that the "extra" SDI frames are almost exactly the same distance apart. The "gaps" are respectively 22340, 22335, and 22334 frames—each about 12 minutes and 25 seconds. In other words, the "extra" frames in the SDI file—or the "missing" frames in the DV file—occur at a regular interval, as though there's some clocking issue or a slight mismatch between frame rates. This doesn't seem to be expected behavior, since we haven't seen it in previous test file sets. The attached zip folder "40000003474808a" contains exports of the "extra" SDI frames, together with exports of the frames to either side of them. The question is whether the "middle" frame in each case is a "real" frame missing from the DV file or a spurious frame that shouldn't be in the SDI file.

40000003823665 (mandolin maker)

This was the second recent reformatting attempt for this tape. Last time, the SDI transfer was 1:05:05 (one hour, five minutes) in length, while the DV transfer was only 1:41 (one min, 41 seconds). In other words, the DV transfer stopped unaccountably a short while into reformatting while the SDI transfer continued for over an hour longer. The new pair of transfers doesn't exhibit this problem—they both continue throughout the whole content of the tape. However, our algorithm did find discrepancies between the DV and SDI files:

DV 1:1301 (0:00:43) > non-consecutive DV timecode/non-consecutive arbitrary bit/
DV 1:1301 (0:00:43) > expected SDI = 1303 (0:00:43); actual SDI = 1304 (0:00:44); dropped frames=1
DV 1:9158 (0:05:06) > non-consecutive DV timecode/non-consecutive arbitrary bit/
DV 1:9168 (0:05:06) > expected SDI = 9171 (0:05:06); actual SDI = 9172 (0:05:06); dropped frames=1
DV 1:13894 (0:07:44) > non-consecutive DV timecode/non-consecutive arbitrary bit/
DV 1:13894 (0:07:44) > expected SDI = 13898 (0:07:44); actual SDI = 13899 (0:07:44); dropped frames=1
DV 1:20700 (0:11:31) > non-consecutive DV timecode/non-consecutive arbitrary bit/
DV 1:20700 (0:11:31) > expected SDI = 20705 (0:11:31); actual SDI = 20706 (0:11:31); dropped frames=1
DV 1:20729 (0:11:32) > non-consecutive DV timecode/non-consecutive arbitrary bit/
DV 1:20729 (0:11:32) > expected SDI = 20735 (0:11:32); actual SDI = 20736 (0:11:32); dropped frames=1
DV 1:20818 (0:11:35) > non-consecutive DV timecode/non-consecutive arbitrary bit/
DV 1:20819 (0:11:35) > expected SDI = 20826 (0:11:35); actual SDI = 20827 (0:11:35); dropped frames=1
DV 1:27834 (0:15:29) > non-consecutive DV timecode/non-consecutive arbitrary bit/
DV 1:27834 (0:15:29) > expected SDI = 27842 (0:15:29); actual SDI = 27843 (0:15:29); dropped frames=1

DV 1:48194 (0:26:48) > non-consecutive DV timecode/non-consecutive arbitrary bit/
DV 1:48194 (0:26:48) > expected SDI = 48203 (0:26:48); actual SDI = 48204 (0:26:48); dropped frames=1
DV 1:60548 (0:33:40) > non-consecutive DV timecode/non-consecutive arbitrary bit/
DV 1:60548 (0:33:40) > expected SDI = 60558 (0:33:41); actual SDI = 60559 (0:33:41); dropped frames=1
DV 1:60757 (0:33:47) > non-consecutive DV timecode/non-consecutive arbitrary bit/
DV 1:60877 (0:33:51) > expected SDI = 60888 (0:33:52); actual SDI = 60889 (0:33:52); dropped frames=1
DV 1:63165 (0:35:08) > expected SDI = 63177 (0:35:08); actual SDI = 63178 (0:35:08); dropped frames=1
DV 1:65629 (0:36:30) > expected SDI = 65642 (0:36:30); actual SDI = 65643 (0:36:30); dropped frames=1
DV 1:65641 (0:36:30) > expected SDI = 65655 (0:36:31); SDI match not detected within 200 frames
DV 1:65641 (0:36:30) > actual SDI = 65654 (0:36:31); backwards match
DV 1:66936 (0:37:13) > non-consecutive DV timecode/non-consecutive arbitrary bit/
DV 1:66938 (0:37:14) > expected SDI = 66951 (0:37:14); actual SDI = 66952 (0:37:14); dropped frames=1
DV 1:67025 (0:37:16) > non-consecutive DV timecode/non-consecutive arbitrary bit/
DV 1:67025 (0:37:16) > expected SDI = 67039 (0:37:17); actual SDI = 67040 (0:37:17); dropped frames=1
DV 1:67054 (0:37:17) > non-consecutive DV timecode/non-consecutive arbitrary bit/
DV 1:67054 (0:37:17) > expected SDI = 67069 (0:37:18); actual SDI = 67070 (0:37:18); dropped frames=1
DV 1:67233 (0:37:23) > non-consecutive DV timecode/non-consecutive arbitrary bit/
DV 1:67233 (0:37:23) > expected SDI = 67249 (0:37:24); actual SDI = 67250 (0:37:24); dropped frames=1

The algorithm found sixteen “drops,” as well as one “backwards match.” Most of the “drops” occur at (or soon after) points where DVAnalyzer reports a timecode anomaly, which is what we’d expect based on our previous experience, and consistent with buffer underruns. But some “drops” don’t fit that profile, particularly in one section around DV frame 65600. This is also where we find the one “backwards match.” At least some of the SDI frames in this section appear to be spurious, like the ones we were finding with the previous test tape. SDI frame 65642 is one example:

The right quarter of the frame has been shifted to the left of the frame and slightly downward, as though the whole image is “off” by one quarter of a scan line. The content of the frame otherwise matches DV frame 65629, and SDI frame 65643 repeats it without the distortion.

There were also a lot of “repeating arbitrary bits,” but by themselves these don’t necessarily raise any red flags for us.

50000000025060 (Test tape with LP/SP transitions)

We created this tape purposefully to test the reformatting system’s ability to handle transitions between SP and LP. It alternates several times between SP and LP, starting with SP.

The SDI transfer contains the whole content of the tape, with both SP and LP sections, but the LP sections are glitchy, with jerky motion and garbled images. The SP sections look fine.

The first DV segment contains the first SP section (roughly frames 0-3742), skips the whole of the first LP section (about 47 seconds), and then continues with the second SP section. This strikes us as unusual; we’d have expected a segment break.

The remaining SP sections on the source tape were captured as separate DV segments, skipping the LP sections.

It looks like the DV stream capture software might be locking into whatever speed it first encounters, with an ability to resume capture at the same speed, but no ability to adjust to a different speed.

In our previous speed-change test, the tape began with an LP section and then went to SP. The SDI transfer in that case was successful throughout (wherever there was “good” content in the first place); we didn’t see any of the glitchiness we’re seeing here with the SDI transfer of LP sections.