Video Reconstruction from Randomized Video Frames

Ned Danyliw
Electrical Engineering
Stanford University
Stanford, CA
Email: edanyliw@stanford.edu

***Abstract*—In certain scientific experiments there is a need to combine sequences of captured static images to reconstruct the dynamics of a system and observe how the system behaves over time. In the case of extracting protein dynamics from many samples, each of which is destroyed after measurement, this is equivalent to reordering a video whose frames have been randomly shuffled. The algorithm proposed uses feature based matching to determine the distance between frames and proposes two sorting algorithms for reordering the frames given the calculated distance metrics. Using these methods, the algorithm achieves good reconstruction of 2D videos.**

I. Introduction

The world around us is constantly in motion and evolving. However when this motion occurs at the smallest scale it is often impossible for researchers to observe. In order to successfully image a system, the sensor needs to capture enough light to fully expose the image fast enough to capture the dynamics of the system. In the case of imaging proteins, like work being done at SLAC National Accelerator Laboratories [1], capturing images requires so much energy that the sample is destroyed in the process. This prevents scientists from capturing any of the dynamics information of proteins such as how they transform between conformations. If many images of the samples are captured, the problem can be viewed as a video reconstruction problem where the frames have been randomly reordered (subject to rotations and repeated frames).

This paper looks at the video reconstruction problem, focusing on the 2D case, with the aim to develop an algorithm that can be generalized to the 3D problem outlined above. The problem was split into two main components - calculating the "distance" between frames to assign likelihoods of frames being adjacent and sorting the frames given these distance metrics. By splitting the algorithm into these two subcomponents, it allows the sorting algorithm to be applied to any set of distance metrics that may be calculated, including distance metrics from the 3D case.

II. Related Work

Reconstructing video from frames that have been randomly shuffled is not a widely researched area. The applications are primarily centered around academic or research communities like those imaging proteins at SLAC. A method that has been gaining traction in the research community has been the Isomap technique proposed by Abbas Ourmazd et al. [2]. This method is a graph-based approach that reconstructs the sequence through a nearest neighbors algorithm after reducing the dimensionality of the system using Isomap. This essentially allows the estimation of the proteins geometry by calculating a lower dimensional encoding of "k" nearest neighbors and then using this greatly reduced dataset for analysis [3]. This reduces the noise in the observed system and performs well when reordering the time-dependent sequence of frames. One disadvantage of this method however is that it "throws away" frames when it reconstructs the sequence from the calculated connected graphs.

Outside of work being done specifically for protein dynamics analysis, predicting video frames is also used in some video compression algorithms [4]. In these algorithms, frames can be reordered and previous and future frames are predicted using motion estimation algorithms. While not immediately applicable to the protein ordering problem, motion estimation techniques may prove useful in increasing the algorithm's robustness to noise and calculating the distance between frames.

III. Algorithm

The algorithm is largely split into three main parts - preprocessing and feature extraction, distance estimation, and sorting. By splitting the reordering algorithm into these smaller steps, it is easier to add and test features as well as generalize the steps to other more complicated applications in the future. The algorithm is implemented in MATLAB and leverages the image processing toolbox for feature extraction and mapping.

A. Feature Extraction and Preprocessing

Before the reordering algorithm is run, the shuffled video data is preprocessed before the features are extracted and then sent to the rest of the algorithm. The distance metrics are calculated solely from the extracted features meaning that the algorithm significantly reduces the dimensionality of the input data through its use of Speeded Up Robust Feature (SURF) feature descriptors. SURF descriptors were chosen because they can be calculated quickly and are robust to translation, rotations, and scaling effects.

The randomized video is first converted to grayscale which is necessary for the SURF feature extraction. Then to speed up the calculation of the features, the images are downsampled

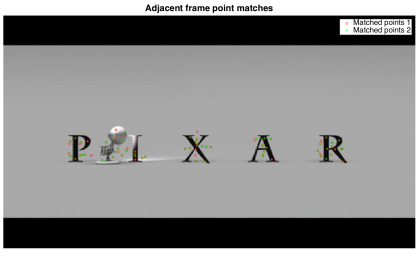

Fig. 1. Adjacent frames with feature matches marked.

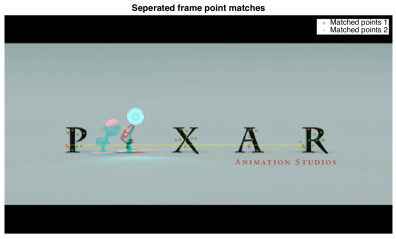

Fig. 2. Separated frames with feature matches marked.

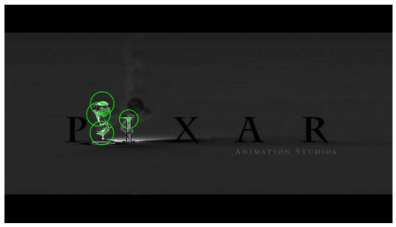

Fig. 3. Demeaned frame with detected features

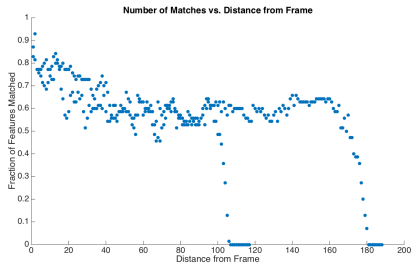

Fig. 4. Number of feature matches vs frame distance

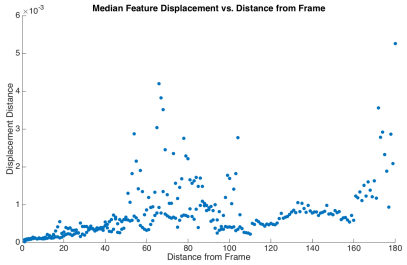

Fig. 5. Median feature displacement vs frame distance

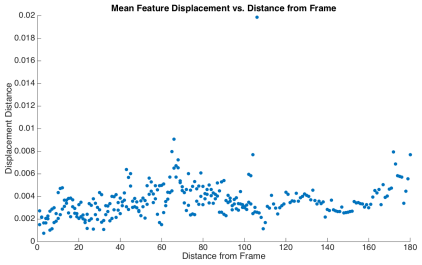

Fig. 6. Mean feature displacement vs frame distance

(used a 16x smaller image for most tests). Finally the features are extracted using MATLAB's built-in SURF detection.

The algorithm also has the option to calculate the features a de-meaned set of the frames. This allows the removal of a static background image from the frames and only detect features on the objects in motion as shown in 3. However in many cases this resulted in too few features being detected to accurately reconstruct the video sequence.

B. Distance calculation

After the features of each frame have been extracted, they are sent to a distance estimation algorithm which quantifies the likelihood of frames being adjacent. This is done by looking at the number of feature matches between frames and the displacement metrics of the matches.

From figures 4-6 we can see that the number of matches is the strongest indicator of the distance between frames. The mean and median displacement of the feature matches can then be used to sort these most likely matches. For most of the test run in this paper, the median was used because of its robustness in the presence of outliers stronger linear relationship to frame distance as shown in figure 5.

For the reordering algorithm, several distance estimation methods were tested. For determining the strongest match available, the distance metric was calculated by ranking the comparisons by number of matches and then sorting the set corresponding to the highest number of matches by their median displacements. This metric performed the best with the "global sort" algorithm which combined the strongest remaining matches until arriving at a final sequence. Meanwhile a matching algorithm that looked at the top 5% of matches and then sorted those by the median displacement metric performed best for the "random seed sort" which builds the sequence linearly from a randomly selected "seed" frame.

An attempt to use machine learning to quantify the weight of each metric in determining the likelihood of frames being adjacent was done with mixed success. Originally a classifier was trained to determine if frames were adjacent but due to the highly skewed set of training examples (only two adjacent frames out of a set of n examples) the classifier had poor

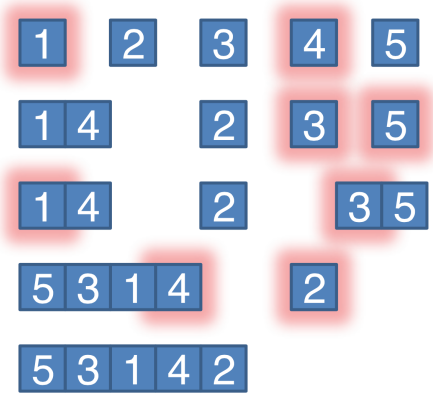

Fig. 7. Illustration of global sort algorithm

performance. When taking into account the probability of being adjacent or fitting a regression to the measured matching metrics the performance was improved but still outperformed by the heuristic distance metrics described above.

C. Sorting Algorithm

Given the distance metrics between frames, the sorting algorithms implemented in this paper attempt to reconstruct the video frames in the most likely sequence. For this two main approaches were taken. The first is a "global sort" that takes into account every possible pairing of frames and sorts by combining the strongest matches until it results in a single continuous sequence. The other sorting algorithm starts from a random frame and attempts to build out by combining the most likely frame that follows in the remaining pool of frames.

1) Global Sort: The global sorting algorithm is done by looking at all the available frames that can be matched, finding the most powerful match, and then combining the associated sequences. This process is illustrated in figure 7.

Figure 7 shows the sorting process. The most powerful matches are highlighted in red. In this example, the algorithm first finds that frames 1 and 4 are the most powerful match and combines those. It then finds the strongest match from the remaining mergeable frames (end frames of sequences and individual frames) and continues merging the strongest matches. When the algorithm matches two heads or two tail frames, it performs the necessary flipping of the sequences to keep the evolution of time correct.

The actual implemented sorting algorithm uses two matrices of the number of feature matches between frames and the median displacement of the matched features. The algorithm then builds the frame groups based on the following pseudo-code

The global sorting algorithm primarily spends its time calculating all the feature matches between each pair of frames which scales as O(N^2). However these calculations can be done in parallel which greatly speeds up the algorithm and the resulting matrices can be sorted very quickly to get the final sequence.

Algorithm 1 Global Sort

Require: X_F = *Matrix of frame matches between frames i and j*
Require: X_M = *Matrix of median displacements between i and j*
Set X_F diagonal to -1 to disallow matching frame with itself
Create *frameGroups* array of grouped frames
while not all frames combined **do**
    $i_m, j_m \leftarrow$ Find the most powerful frame match
    if i_m, j_m in same group **then**
        Set $X_F(i_m, j_m) = -1$
    else
        Combine frame groups containing i_m, j_m
        if i_m, j_m internal **then**
            Set corresponding rows and columns to -1
        end if
    end if
end while

2) Random Seed Sort: The random seed sort algorithm reduces the number of calculations by building the sequence from a randomly selected seed frame instead of searching over the entire space of available frames. The algorithm builds out the frames in a single direction, picking the best available next frame from the available pool of frames until it determines it it better to build in the opposite direction.

The sorting algorithm is shown in figure 8. In this example, frame 4 is selected as the seed frame. Then the algorithm builds off in one direction combining with the best match in the available pool. This process continues until the best match it selects turns out to be closer to the opposite end of the constructed sequence. At this point the algorithm returns the frame to the pool, reverses the build direction and continues. This process continues until all the frames have been merged.

This process can be expressed by the following psuedocode

Algorithm 2 Random Seed Sort

Create *frameGroups* array of grouped frames
Set *head* and *tail* to a random seed frame
while not all frames combined **do**
    if building from *tail* **then**
        toMatch $\leftarrow$ *tail*
        altMatch $\leftarrow$ *head*
    else
        toMatch $\leftarrow$ *head*
        altMatch $\leftarrow$ *tail*
    end if
    $i \leftarrow$ best match with *toMatch*
    if i is closer to *altMatch* **then**
        Return i to available pool
        Flip build direction
    else
        Append or prepend i to *toMatch*
    end if
end while

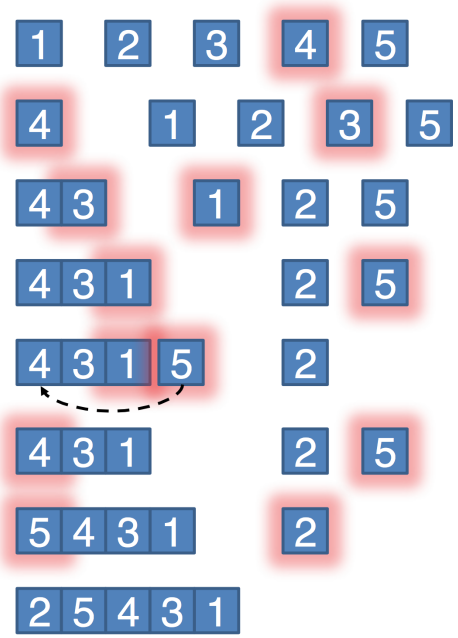

Fig. 8. Illustration of random seed sort algorithm

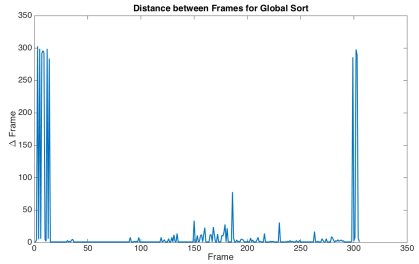

Fig. 9. True frame distance versus frame ordering for global sort

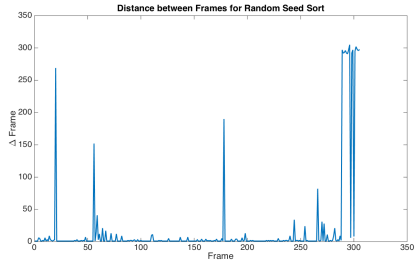

Fig. 10. True frame distance versus frame ordering for random seed sort

This algorithm has the advantage of building the sequence immediately and requiring fewer comparisons however it suffers from compounding errors. For example if the sort picks the wrong frame for i, then future sorting will be done from the incorrect frame resulting in occasional sequence jumps.

IV. Results

A. Test Videos

To test the reordering algorithm, several shuffled videos were used as inputs. The key requirement was that the chosen videos don't contain scene jumps which the algorithm would have no way of reconstructing (and wouldn't be present in experimental data). The chosen videos were the Pixar intro, a rotating and transforming cube, and a video of a football throw.

B. Measuring Performance

While it is fairly easy to qualitatively evaluate if a video has successively been reordered, it is more difficult to quantify the performance of the algorithm. This is because it is not expected that the algorithm perfectly reconstructs the original video since it has no knowledge of the direction of time or start of the video. However it is desired that reconstructed adjacent frames are close to the true adjacent frames of the original video. When designing the performance metric, it should not penalize the algorithm too heavily if the reconstruction's adjacent frames are close to the frame. Additionally it should not be overly penalized if it selects a frame that is very far away versus one that is far away. The justification for this is that it is just as bad to jump 30 frames as it is to jump 300.

All of these requirements led to the choice of a logistic performance metric. Then the function can be characterized by the function's offset and scaling factor. The score for frames f_1 and f_2 can be given by

$$l(f_1, f_2) = \frac{1}{1 + \exp(-0.5(\text{abs}(f_1 - f_2) - 10))}$$

The overall performance is evaluated by summing all these metrics for adjacent frames and normalizing by the total number of frames. The lower the metric the better the match.

One shortcoming of this metric is that it doesn't scale perfectly when considering very few frames (the weight of one mismatch is much higher). However between videos of the same length it largely agrees with a qualitative evaluation of the algorithm. One notable exception is in videos with several large sequence jumps in the reconstructed video. Because the algorithm isn't penalized too much for these jumps (they are only one sample) a video with multiple jumps may score better than one that has a higher average mismatch for a period of time. This doesn't necessarily correspond to what is "visually pleasing" but is arguably a better metric for overall performance.

C. Input Image Scale

The algorithm was evaluated on how it performs when changing the scale of the input images. By reducing the size of the images used for feature detection, the initial feature detection and feature matching runs faster and with a lower memory footprint, something important for much larger datasets. The results shown in figure 11 showed that the global sorting algorithm had more consistent results while the random seed sort benefited greatly from larger input image sizes.

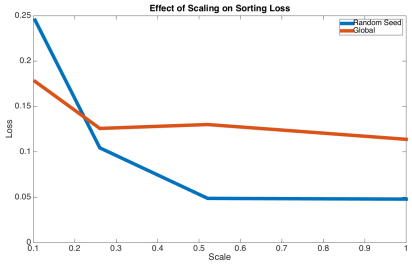

Fig. 11. Algorithm performance with varying input image scales

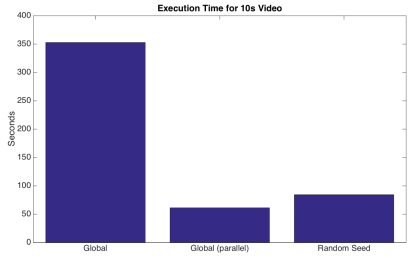

Fig. 12. Algorithm execution times for a 10s input video

D. Execution Time

The random seed sort executes much faster than the global sort (about 4x faster). However the global sort's execution time is largely determined by the feature match calculations which can be parallelized. Then the feature matching time drops from 348.3s for a 10s video to 56.8s (The sort time stays at 4.5s). Parallelization is more difficult with the random seed algorithm so the global sort may be in fact faster for most systems if it can be run in parallel.

E. Frame Skipping

To test how the reordering algorithm handles larger distances between frames, the algorithm was evaluated with skipping frames in the input video before shuffling. Again the global sort proved to be more robust to larger differences between frames. One thing to note about these results is that they highlight some of the shortcomings of the fit metric that was used. Because there were fewer frames in the largely downsampled versions and the loss metric requires more than about 20 frame mismatch for the max penalty, the loss figures are lower than the quality of the video would suggest.

F. Rotation

The reordering algorithm's performance was also evaluated when the input images are randomly rotated before the features are detected. The algorithm performed significantly worse (0.706 for the global sort and 0.845 for the random seed sort with 5% random rotations). This seems to be due to the distance metrics used and would need to be improved before the algorithm can be used in the presence of rotations.

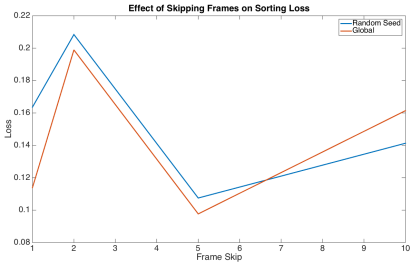

Fig. 13. Algorithm performance when skipping frames

V. Conclusion

This algorithm shows that randomized video frames can be accurately reordered using feature-based matching and then passing the calculated distance metrics into either the global or random seed sorting methods. The global sort provides a more robust ordering that can be greatly accelerated though paralleling the calculation of the distance metrics. However the random seed algorithm shows that shorter sequences can be effectively ordered if more frequent sequence jumps are tolerable.

Due to the decoupled nature of the distance metric calculation and the sorting algorithm, the approaches described in this paper can be combined with other algorithms that may be more appropriate for the final application. For example the distance metric can be replaced with a 3D variant like RMSD (root-mean-square of atomic deviations) which is often used in protein research. This metric can then be used to sort the captured frames using the global sort algorithm.

VI. Future Work

There are several areas that the proposed algorithm can be improved and better suited for generalizing to the protein dynamics reconstruction problem. The first would be to experiment with using the sorting algorithms with different distance metrics. This would involve improving the distance metrics used with 2D images to make them more robust towards rotations and determining the optimal weightings for calculating this distance metric from the feature matches. Along with improving the distance metric calculation, the sorting algorithm can be further optimized. Some ways it can be improved is by designing a random seed sort that behaves more like the global sort by building multiple groups of frames simultaneously and possibly caching comparisons to speed up execution. Finally the true test for this algorithm would be to try applying the approaches used in this paper to sort experimental data from SLAC and reconstruct a 3D protein's dynamics.

Acknowledgment

The author would like to thank the EE368 course staff and especially TJ Lane for their support and guidance on this project.

REFERENCES

[1] https://portal.slac.stanford.edu/sites/lcls_public/Pages/Default.aspx

[2] P. Schwander, D. Giannakis, C. H. Yoon, and A. Ourmazd, *The symmetries of image formation by scattering. II. Applications*, Optics Express, 4 June, 2012.

[3] J. Tenenbaum, V. Silva, and J. Langford, *A Global Geometric Framework for Nonlinear Dimensionality Reduction*, Science, 22 December, 2000.

[4] B. G. Haskell and A. Puri, *MPEG Video Compression Basics*, Chapter 2, 25 August, 2011