electronics

MDPI

Article

FlexAlign: An Accurate and Fast Algorithm for Movie Alignment in Cryo-Electron Microscopy

David Střelák [1,2,3,*], Jiří Filipovič [2], Amaya Jiménez-Moreno [3], Jose María Carazo [3] and Carlos Óscar S. Sorzano [3]

1 Faculty of Informatics, Masaryk University, Botanická 68a, 60200 Brno, Czech Republic
2 Institute of Computer Science, Masaryk University, Botanická 68a, 60200 Brno, Czech Republic; fila@mail.muni.cz
3 Spanish National Centre for Biotechnology, Spanish National Research Council, Calle Darwin, 3, 28049 Madrid, Spain; ajimenez@cnb.csic.es (A.J.-M.); carazo@cnb.csic.es (J.M.C.); coss@cnb.csic.es (C.Ó.S.S.)
* Correspondence: dstrelak@cnb.csic.es

Received: 14 May 2020; Accepted: 18 June 2020; Published: 23 June 2020

Abstract: Cryogenic Electron Microscopy (Cryo-EM) has been established as one of the key players in Structural Biology. It can reconstruct a 3D model of the sample at the near-atomic resolution, which led to a *Method of the year* award by Nature, and the Nobel Prize in 2017. With the growing number of facilities, faster microscopes, and new imaging techniques, new algorithms are needed to process the so-called movies data produced by the microscopes in real-time, while preserving a high resolution and maximum of additional information. In this article, we present a new algorithm used for movie alignment, called FlexAlign. FlexAlign is able to correctly compensate for the shift produced during the movie acquisition on-the-fly, using the current generation of hardware. The algorithm performs a global and elastic local registration of the movie frames using Cross-Correlation and B-spline interpolation for high precision. We show that our execution time is compatible with real-time correction and that we preserve the high-resolution information up to high frequency.

Keywords: cryo-em; movie alignment; acceleration; gpu; flexalign; cuda; autotuning; cufft; cufftadvisor

1. Introduction

The processing pipeline of the Cryo-EM consists of several steps, movie alignment being the very first one. A movie is a sequence of frames produced by the microscope, with each frame recording a projection of tens to hundreds of particles. By averaging frames, a micrograph is produced, which is later used for particle picking, Contrast Transfer Function (CTF) estimation, and other steps of the image processing pipeline. Due to beam-induced motion and other changes within the recorded area during the screening, simple averaging of the frames is not sufficient.

To obtain a single frame, a beam of electrons is fired against the sample. After passing the sample, the beam is recorded by a direct electron detection camera. This is known as Transmission Electron Microscopy (TEM), see Figure 1. Since the electron beam causes radiation damage, the electron dose has to be very low, about one electron per Å^2 and frame (electron arrival is supposed to occur following a Poisson distribution; this means that the most common observations are 0 or 1 electron per pixel). This, on the other hand, results in an extremely low Signal-to-Noise Ratio (SNR). To get more signal, we can repeat the imaging several times using the same sample and then average the resulting frames.

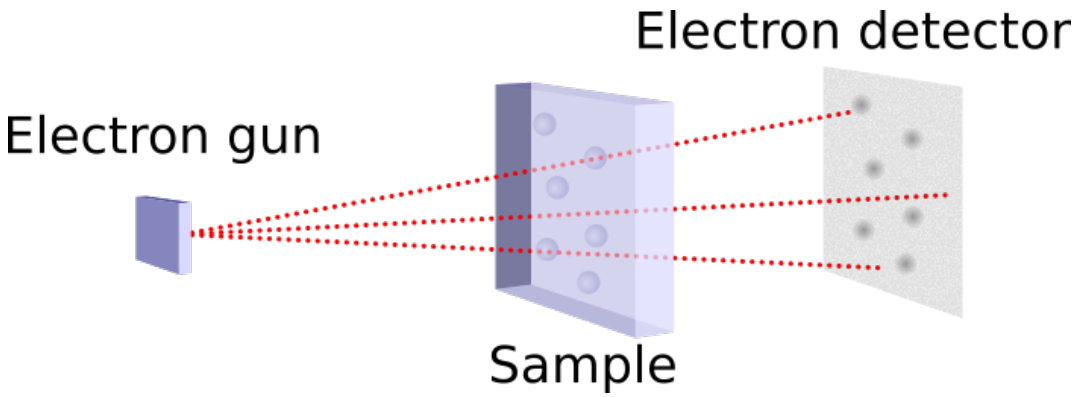

Figure 1. Principle of the TEM.

However, before averaging the frames, they have to be properly aligned as the sample moves during the acquisition. The reasons for this movement may vary from sample to sample and they are carefully described by [1]. This movement can be both global and local, and both types need to be corrected.

Global alignment is trying to compensate for the apparent movement of the entire frame. Even though this can lead to incorrect alignment at a specific location, the overall SNR will be highly improved. For that reason, it is often used as the first step before local alignment. In Figure 2, we can see the possible effect of (not) performing the global alignment on a noiseless phantom movie, generated and aligned as described in Section 4.1.

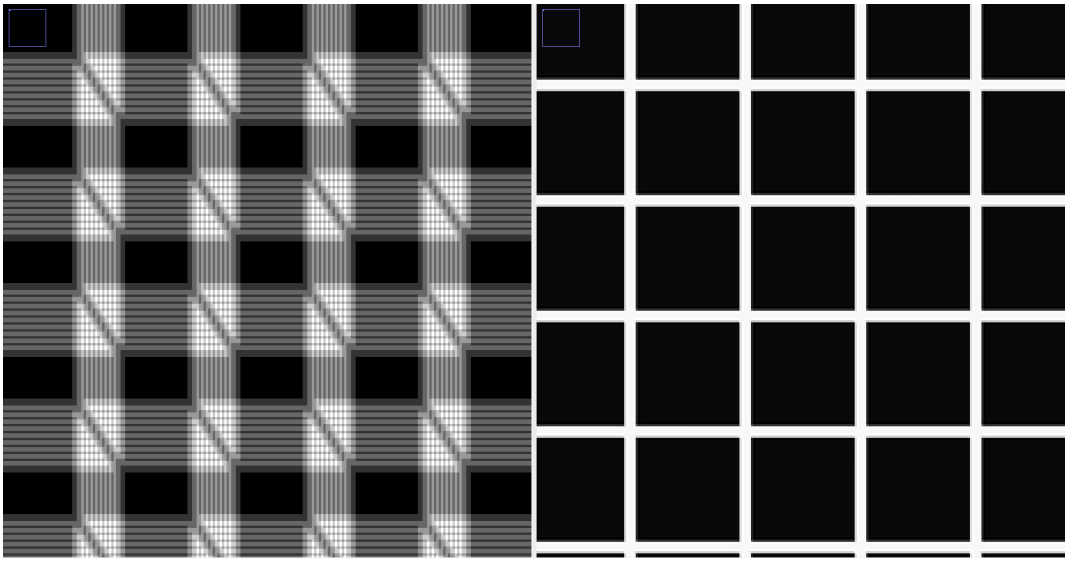

Figure 2. Phantom movie (grid, detail), an average of 10 frames, nth frame shifted by the vector $[2n, 3n]$, before global alignment (**left**), after global alignment (**right**).

The aim of the local alignment is to compensate for more complex movements of the particles, should they be caused by the beam, doming, or another cause. Typically, it works on a divide and conquer basis—the movie is divided into small patches, and the alignment is solved independently for each patch. Figure 3 shows the possible effect of (not) performing the local alignment on a phantom movie.

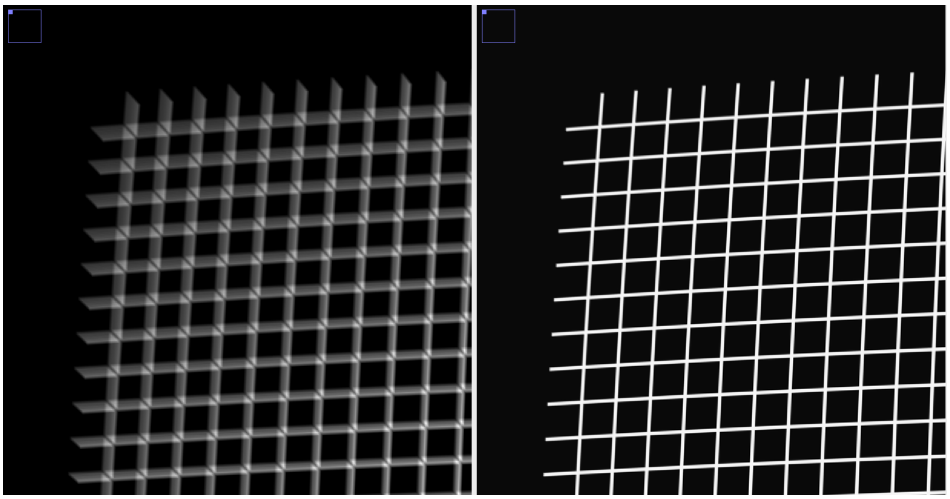

Figure 3. Phantom movie (grid, detail), an average of 50 frames, frames shifted + doming applied, using only global alignment (**left**), after local alignment (**right**).

The requirement for fast and precise algorithms for movie alignment is driven by three factors. The first is speed. The new generation of detectors [2] and acquisition practices reduced the recording time to 4 movies per minute [3], and this time is expected [4] to reach as little as 5 s per movie soon. It is crucial to be able to process these movies in real-time, as potential problems with the imaging can be corrected as soon as they appear during the acquisition (the access cost to the microscope ranges from $1000 to $3000 per day).

The second is accuracy. The goal of Cryo-EM is to produce near-atomic models of the macromolecules under study [5]. This goal sets an important challenge to all the image processing steps, especially this one, as the SNR of the micrographs ranges from 1/10 to 1/100 (At the level of the frame, this SNR has to be divided by the number of frames, typically between 10 and 100.).

The third is particle tracking for polishing. Being able to accurately track the particles back to the originating frames is crucial during the polishing phase, which aims to further improve the resolution of the final 3D reconstruction.

In this article, we introduce a tool for movie alignment called FlexAlign. We give details of our algorithm to perform the movie alignment using the Graphical Processing Unit (GPU) and Compute Unified Device Architecture (CUDA). We propose a two-stage (global and local) movie alignment algorithm based on Cross-Correlation (CC) and B-spline interpolation for the description of the local shifts. As we show, FlexAlign produces high contrast micrographs, it is rather robust to noise, and it is able to process movies at the microscope acquisition speed, using the current generation of the GPUs.

The rest of the paper is organized as follows. Section 2 gives additional details on movie alignment, including the (non) functional requirements of the algorithm. In Section 3, we describe our implementation. Quality and performance evaluation is done in Section 4. Conclusions and future work can be found in Section 5.

1.1. Comparison to Other Implementations

Probably the most commonly used SW for movie alignment is currently MotionCor2 [6]. While MotionCor2 provides good performance and precision, it is not providing, to the best of our knowledge, the data needed for particle tracking. It allows for both global and local alignment and is accelerated on GPU. Similarly to our method or the one of Warp [7], it uses CC to align frames or patches of the movie.

The Optical Flow [8] approach can describe the per-pixel movement, but at the cost of high storage requirements—for each pixel of each frame, a 2D shift-vector has to be stored (Provided the shift-vector is stored in single precision, we would need 128 MB per frame of 4000 $\times$ 4000 pixels, or 6.25 GB for a movie with 50 frames. Movies are often stored in single-precision, uncompressed format, meaning

that the storage requirement would triple.). In addition, the optical flow is computationally expensive, even in a GPU accelerated version.

Relion [9] implements Bayesian polishing, and to expose the metadata of the movie alignment, they provide their Central Processing Unit (CPU) version of the MotionCor2.

The brief overview of the current alignment algorithms can be found in the Table 1.

The goal of FlexAlign is to combine the best of these, namely, the short computational time, the flexibility of elastic deformations, support for detailed pixel tracking, and open-source implementation.

Table 1. Comparison of various movie alignment algorithms.

Program	HW	Method + Interpolation
MotionCor2	GPU	CC + polynomial
Relion MotionCor	CPU	CC + polynomial
Optical flow	CPU/GPU	optical flow + cubic interpolation
Warp	GPU	CC + higher-order schemes
FlexAlign	GPU	CC + B-spline

2. Movie Alignment

2.1. Our Method

2.1.1. Global Alignment

For each pair of frames f_i and f_j (where $j > i$), we estimate their apparent shift $\hat{\mathbf{r}}_{ij}$ exploiting the correlation theorem of the Fourier Transformation (FT). Then, we try to find a sequence of shifts between one frame and the next that explains the observed shifts between any pair of frames (see Equation (1)). For doing that, we try to find a unique shift-vector per frame such that the sum of the shifts of all intervening frames $\mathbf{r}_i \dots \mathbf{r}_j$ matches the observation:

$$\mathbf{r}_i + \mathbf{r}_{i+1} + \cdots + \mathbf{r}_{j-1} = \hat{\mathbf{r}}_{ij} \tag{1}$$

We get an overdetermined set of linear equations, as the number of unknowns $\mathbf{r}_1 \dots \mathbf{r}_{n-1}$ is smaller than the known shifts $\hat{\mathbf{r}}_{ij}$. It can be expressed in a matrix form as

$$A\mathbf{r} = \hat{\mathbf{r}} \tag{2}$$

The actual per frame movement can be computed by solving Equation (3) in the Least Squares sense

$$\mathbf{r} = (A^T A)^{-1} A^T \hat{\mathbf{r}} \tag{3}$$

The equation system in Equation (2) may contain outliers due to misestimates of the true shifts between pairs of frames. After solving the equation system, we compute the residuals for each equation and remove those equations whose residual is larger than three standard deviations in absolute value. Then, the equation system is solved again with the remaining equations.

For each frame within the micrograph, we report the global shift parameters. In this way, decisions on the stability of the acquisition process can be quickly taken.

2.1.2. Local Alignment

The local alignment uses a principle similar to the global alignment. We cut each frame into several possibly overlapping rectangular segments (see Figure 4). The equivalent segments in consecutive frames are called patches.

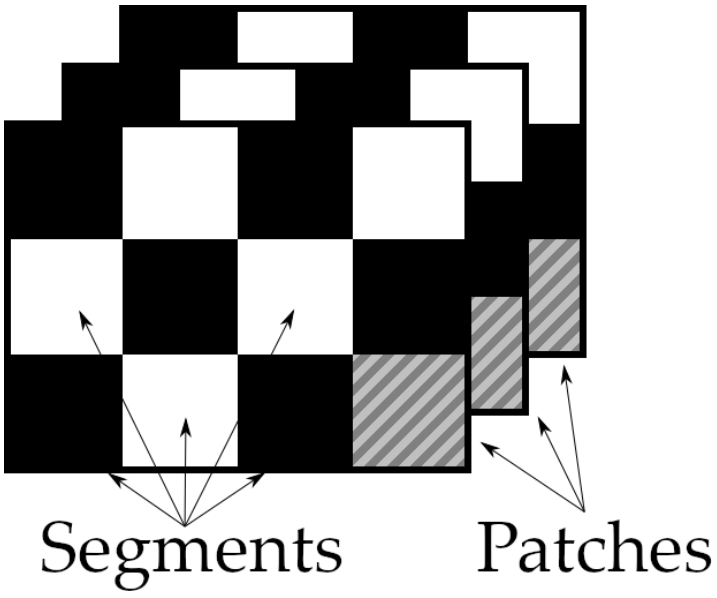

Figure 4. Division of the frames to patches for local alignment.

Patches can be handled in the same way as regular movie frames, i.e., we can compute the alignment between them. Once we know the relative shift between all pairs of patches, we can try to fit a smoothing function to this data. We use cubic B-splines, which provide a good trade-off between smoothing quality and computational complexity [10]. Let us refer as $\mathbf{r}_i^p = (r_{i,x}^p, r_{i,y}^p)$ to the shift of the patch p in frame i calculated in the same way as we did for the global alignments. Let us refer as (x_p, y_p) to the coordinate center of the patch p in the coordinate system of the micrograph ($(0,0)$ is in the top-left corner) before alignment. Then, we look for the B-spline coefficients $c_{lmn,x}$ and $c_{lmn,y}$ such that

$$r_{i,x}^p = \sum_l \sum_m \sum_n c_{lmn,x} B\left(\frac{i}{T_f} - l\right) B\left(\frac{x_p}{T_x} - m\right) B\left(\frac{y_p}{T_y} - n\right) \tag{4}$$

where $B(\cdot)$ is the cubic B-spline function, T_f, T_x, and T_y are the separation between spline nodes in the time, x and y directions, respectively. The equation above should hold for all patches p and frames i. A similar equation applies to $c_{lmn,y}$. Note that the equation system above is linear in the c coefficients, and we also solve it through the Least Squares problem in which we identify outliers and remove them.

We store the B-spline interpolation c coefficients together with the interpolation nodes geometry (T_f, T_x, and T_y), with each micrograph. In that way, we are able to backtrack the movement of each pixel of each frame and allow for particle polishing (that is, a more precise local shift estimation per particle).

2.1.3. Complexity

As can be seen, for the global alignment, our algorithm needs N forward FTs, and $\frac{N(N-1)}{2}$ inverse FTs, where N is the number of frames. The dimension of the data during the forward FT is typically the original size of the frame, while the size of the inverse FT is usually smaller (Full-scale forward FT is followed by a cropping of the high-frequency data. While downscaling speeds up the transformation, it also improves the precision by suppresing noise.).

The local alignment part needs an additional MN forward FTs and $M\frac{N(N-1)}{2}$ inverse FTs, where M is the number of segments.

Other operations, such as equation system solving, are negligible in comparison to the total computational demand of the FTs.

3. Implementation

3.1. Alignment Estimation

The pseudocode of the core functionality—alignment estimation of several images (either frames or patches) is shown in Algorithm 1 and Figure 5. As can be seen, for each image, we need to iterate through all consecutive images and compute the relative shift using CC. Ideally, both parts of the algorithm would be executed on GPU, to avoid memory transfer overhead. However, the current generation of GPUs is still manufactured with a rather small amount of on-board memory, which is typically in the range of 6–12 GB on consumer-class cards, and up to 48 GB on professional-class cards.

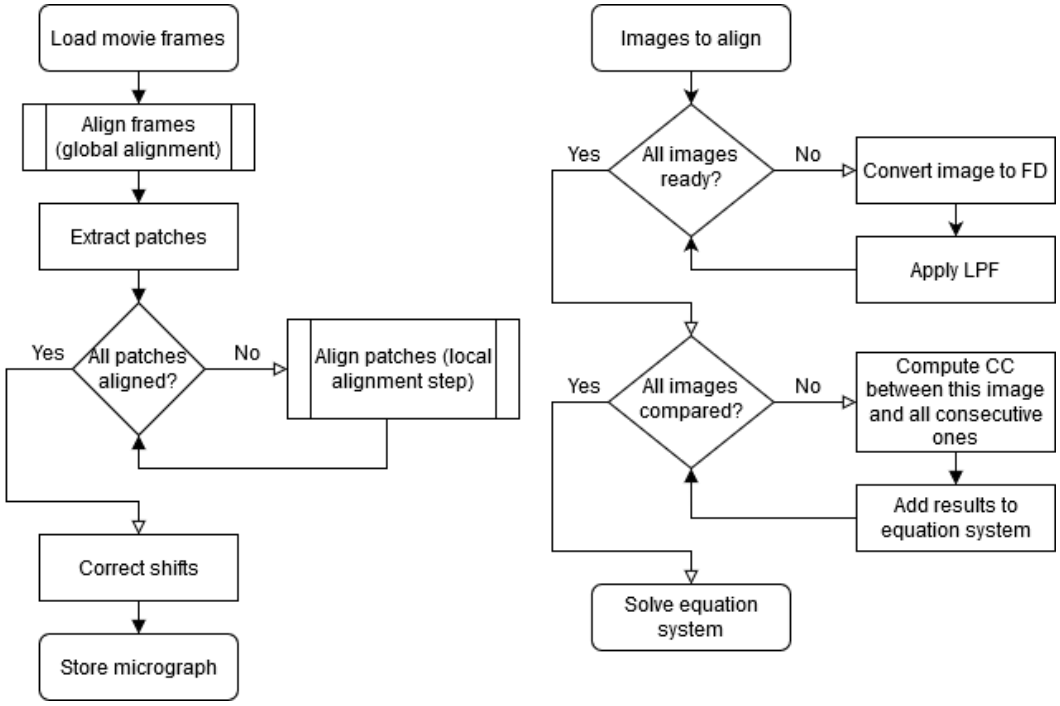

Figure 5. Flowchart of the core idea of the algorithm. Algorithm overview (**left**), alignment subroutine (**right**).

Algorithm 1 Compute alignment estimation

function COMPUTE_ALIGNMENT($images$) // frames or patches
  $images'_c = \{\}$
  for all $img_i \in images$ **do**
    $img'_i = FT(i_1)$ // conversion to FD
    // crop very high frequencies and apply the filter on the rest
    $img'_{ic} = lowpass\_filter(img'_i, filter)$
    $images'_c.append(img'_{ic})$
  $solver = equation\_system\_solver()$
  for all $img'_{ic} \in images'_c$ **do**
    for all $img'_{jc} \in (img'_{(i+1)c}, \ldots img'_{nc})$ **do**
      $x, y = shift\_alignment(img'_{ic}, img'_{jc})$
      $cc' = i''_{1c} \cdot i''_{2c}$ // cross-correlation
      $cc = IFT(cc')$ // inverse transformation
      $x, y = pos(max(cc))$ // position of the maxima is the estimated shift
      $solver.add\_equation((img'_{ic}, img'_{jc}), (x, y))$
  return $solver.solve()$

Like other authors, we have used batch processing to overcome this issue. This requires the reorganization of some steps of the algorithm. First, we transfer N images to GPU, perform forward

FT, low pass filtering including cropping, and we download the resulting images in the Frequency Domain (FD) to Random Access Memory (RAM) (see Line 7 of Algorithm 1). N is selected such that we have enough space for out-of-place FT, cropped data and the low pass filter. Transfer back to RAM creates natural synchronization/debug point, and allows us to process data even on low-end GPUs, or high-resolution movies with many frames. The pseudocode of this process is shown in Algorithm 2. For simplicity, we skip the boundary and other checks and kernel settings.For filtering, we use standard low-pass Gaussian filter with given maximal resolution. Due to the properties of the Gaussian distribution (99.7% of the values are within three standard deviations from mean.), we know that values at four standard deviations are almost zero, i.e., we can crop out these values completely. The remaining values are multiplied by the weight as given by the distribution. We compute sigma as $\sigma = \frac{T_s}{R_{max}} \cdot \sqrt{\frac{-1}{2\log 0.5}}$, where T_s is the target resolution in pixels and R_{max} is the maximal resolution to preserve in Å.

The following step (of Algorithm 1) is the computation of the CC between images. The pseudocode of this process, without boundary and other checks, is outlined in Algorithms 3 and 4.

Algorithm 2 Batched FT and low-pass filtering

```
1:  function TRANSFORM_TO_FD(images, filter)
2:      result = {} // cropped and filtered images in FD
3:      N = find_batch_size(images, filter)
4:      for i = 0; i < |images|; i += N do
5:          batch = transfer_to_GPU(images, i, i + N)
6:          batch' = FT(batch) // conversion to FD
7:          batch'_filtered = lowpass_filter_kernel(batch', filter)
8:          result.append(transfer_from_GPU(batch'_filtered))
9:      return result
```

Algorithm 3 Batched shift estimation

```
1:  function COMPUTE_SHIFT_ESTIMATION(images)
2:      result = {} // cropped correlation centers
3:      I, J = find_buffer_and_batch_size(images)
4:      for i = 0; i < |images|; i += I do
5:          buffer1 = transfer_to_GPU(images, i, i + I)
6:          result.append(batch_cc(buffer1, buffer1, J) // see Algorithm 4
7:          for j = i + 1; j < |images|; j += I do
8:              buffer2 = transfer_to_GPU(images, j, j + I)
9:              result.append(batch_cc(buffer1, buffer2, J) // see Algorithm 4
10:     return result
```

Algorithm 4 Batched CC

```
1:  function BATCH_CC(buffer1, buffer2, batch_size)
2:      result = {}//cropped correlation functions
3:      no_of_cc = compute_number_of_correlations(buffer1, buffer2)
4:      for i = 0; i < no_of_cc + batch_size; i += batch_size do
5:          cc' = pointwise_multiply_and_shift_kernel(buffer1, buffer2, i)
6:          cc = IFT(cc')
7:          centers = crop_centers(cc)
8:          result.append(transfer_from_GPU(centers))
9:      return result
```

We use four buffers on the GPU:

- The first two represent two floating windows, each holding I consecutive images from the previous step.
- The other two hold results of J pointwise multiplications of the images, in FD and in Spatial Domain (SD) respectively (variables cc and cc' on Lines 5 and 6 of the Algorithm 4).

I and J are selected so that they fit into available GPU memory. First, we upload to GPU I images into the first buffer, and another I images to the second buffer. Then, we run a kernel performing pointwise multiplication of all pairs of images in the first buffer, in batch of J images (see the first step in Figure 6). Since we always use images of even size, we also multiply each element of the resulting image by $-1^{(i+j)}$, where i, j are positions of each element. After inverse FT, this results in shifting the correlation function to the center of the image. The last step is to crop the correlation centers and download them to RAM for further processing.

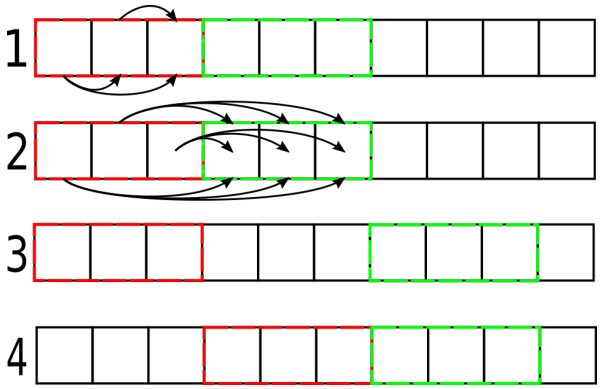

Figure 6. Data processing using two buffers. First, we process images in the first (red) buffer (**1**), then all pairs of images in both buffers (**2**). We iteratively fill the second (green) buffer with remaining images (**3**, arrows skipped for brevity). When all images pairs for the first buffer are processed, we load new images into the first buffer (**4**).

Similarly, we get centers of correlations between images in the buffers, see the second step in Figure 6. We keep loading consecutive images to the second buffer until we pass through all of them, as shown in step three in Figure 6. Once that happens, we upload new images also to the first buffer and repeat the process until all pairs of images are processed.

To locate the position of the maxima, we use sub-pixel averaging. We locate the pixel with maximal value. From it, we search for the nearest pixel with value $max/\sqrt{2}$. Distance to this pixel determines an area, in which we do a weighted average of the pixels. The position of this average is the requested shift. Due to a complicated memory access pattern and low exposed parallelism, we kept this code on the CPU side. The obtained positions of the maxima, i.e., relative shifts of all pairs of frames, are used as input to the linear equation system solver, as described in Section 2.1.1.

3.2. Global Alignment

Our algorithm starts by loading all frames into a consecutive block of RAM. During the load, we also apply a gain and dark image correction (A dark image is a residual signal generated by the sensor when no electrons are fired at it, and a gain image corrects uneven sensitivity of the sensor's pixels.), if requested. Then, we apply a low pass filter and estimate the shifts of all frames, as described in the previous subsection.

3.3. Local Alignment

Local alignment is a direct extension of the global alignment. We use the fact that we already have frames loaded in RAM. We process the frames by segments—we copy out proper patches, and at the same time, we compensate for the global shift estimated in the previous step. Since patches are typically smaller than frames, we average data from several (three, by default) frames together, to suppress the noise and enhance the signal. As typical particle has size of 200–300 Å, we use by default patches of 500 × 500 Å, so that at least one particle can fit in it.

Once extracted, patches are then handled in the same fashion as frames of the global alignment. The pseudocode without frame averaging is shown in Algorithm 5.

Algorithm 5 Local movie alignment

```
function LOCAL_ALIGNMENT(frames)
    global_shift = compute_alignment(frames)
    patch_shifts = {} // absolute shift of each patch
    for all s ∈ segments do
        solver = equation_system_solver()
        patches = extract_patches_from_frames(frames, global_shift, s)
        patches_shifts = compute_alignment(patches)
        solver.add_equations(patches, patches_shifts)
        patch_shifts.append(s, solver.solve())
    return patch_shifts
```

3.4. CuFFTAdvisor

As can be seen from the text above and Section 2.1.3, our method is strongly dependent on the performance of the FT.

We use the standard NVIDIA's *cuFFT* library [11]. As the performance and additional memory consumption of this library depend on the size of the input data and type of the transformation, we have developed custom software, cuFFTAdvisor [12] that uses autotuning to help us to automatically determine the best settings.

During the global alignment, we search for the size of the frame, which could be up to 10% smaller and is faster to process. This can also result in up to 8 times less space used by the library itself, see [12] for details.

As stated earlier, during filtering, we crop high frequencies (over four standard deviations) of the frames in the FD. We autotune for the sizes which could be up to 10% bigger, but faster to process. Once we know this size, we simply have to alter the filter to take this size change into account.

Sizes of the (filtered) patches during the local alignment are obtained in a similar fashion.

To compensate for the time spent on autotuning, we store the autotuned sizes in a user-defined file as key-value pairs. As key, we use a combination of the GPU used, input size of the movie, and available GPU memory. The value is, then, the optimized size.

Users can also opt-out of this autotuning step, should it lead to overall slow-down, e.g., during the processing of a few movies only.

4. Quality and Performance Analysis

To ensure that our algorithm is able to cope with complex movements and noise, we have run a collection of tests.

4.1. Phantom

First, we have generated a phantom movie, consisting of 30 frames of 4096 × 4096 pixels. Each frame contains a grid of 5 pixels wide lines 50 pixels apart. Grid pixels were set to one, background ones to zero. Each pixel $[x, y]$ of each frame t has been shifted using the following formulas:

$$\begin{aligned} x(t) &= a_1(n-t) + a_2(n-t)^2 + cos(n-t)/10 \\ y(t) &= b_1(n-t) + b_2(n-t)^2 + sin((n-t)^2)/5 \end{aligned} \tag{5}$$

After the shift, a doming has been applied, with the center of the doming located in the middle of the frame, using the normalized distance r from it:

$$\begin{aligned} k_1 &= k_{1s} + t(k_{1e} - k_{1s})/n \\ k_2 &= k_{2s} + t(k_{2e} - k_{2s})/n \\ r_{out} &= r_{in}(1 + k_1 r_{in}^2 + k_2 r_{in}^4) \end{aligned} \tag{6}$$

where r_{out} and r_{in} are the radial coordinate of the pixel in the output and input images, respectively.

We have used the following constants: $n = 30; a_1 = -0.039; a_2 = 0.002; b_1 = -0.02; b_2 = 0.002; k_{1s} = 0.04; k_{1e} = 0.05; k_{2s} = 0.02; k_{2e} = 0.025$ and bilinear interpolation between pixels.

We got a phantom movie with a grid, which has both translation and doming applied between each frame. We used this phantom to simulate the image acquisition process in the electron microscope as follows. For each pixel value v, we simulate the arrival of i electrons using a Poisson distribution whose average is controlled by the corresponding phantom pixel:

$$\begin{gathered} \mu = \mu_b - (\mu_b - \mu_f)v \\ P(i|\mu) = \frac{e^{-\mu}\mu^i}{i!} \end{gathered} \tag{7}$$

We used values $\mu_b = 1.25$ for background pixels and $\mu_f = 1.05$ for foreground (grid) pixels.

Using this phantom, we have tested FlexAlign, MotionCor2 v.1.3.1, Warp 1.07W, cryoSPARC v2.15 (patch_motion_correction_multi) [13] and Relion's MotionCor v.3.0.8. We have used 9×9 segments per frame of 500×500 pixels (assumed pixel size = 1.0). CryoSPARC sets the number of segments to 7×7 and does not allow for manual override. As can be seen in Figure 7, both MotionCor2 and FlexAlign were able to correct combined shifts correctly, FlexAlign producing a higher contrast micrograph. High contrast is important especially during the picking stage, where it allows for more reliable particle selection, and also could indicate better high-frequency preservation. The other three programs were not able to correct for the shift near the corner of the resulting micrograph. While Relion MotionCor and cryoSPARC get gradually worse, Warp produces a sharper transition.

4.2. Real Movies

We have run similar tests on experimental data. We used the EMPIAR data sets 10288 [14], 10314 [15] and 10196 [16].

4.2.1. EMPIAR 10288

The EMPIAR 10288 dataset consists of movies of 40 frames, each frame having a resolution of 3838×3710 pixels. The pixel size is 0.86 Å, which resulted in 9×9 patches. The average exposure and the camera model is not specified. Gain correction images are provided for a subset of the movies.

Figure 8 shows a trace of the reported global alignment and histogram of the first frame. As we can see, all programs report very similar shifts, except Warp, which reports around two pixels shorter track. It is worth mentioning that this might be compensated during the local alignment step.

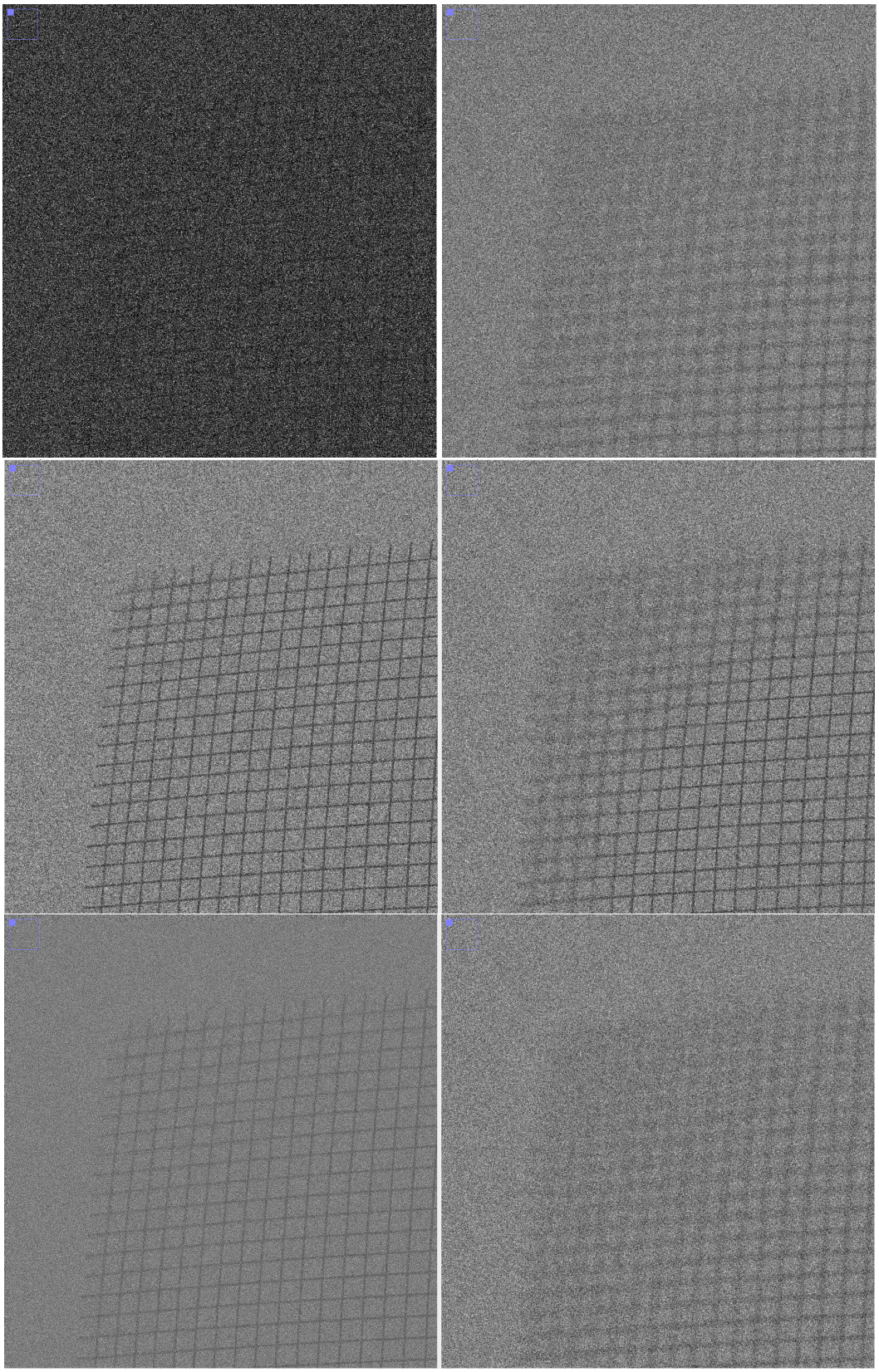

Figure 7. Detail of the frame of the phantom movie (**top left**), detail of produced micrograph, normalized: cryoSPARC (**top right**), FlexAlign (**center left**), Warp (**center right**), MotionCor2 (**bottom left**), Relion MotionCor (**bottom right**).

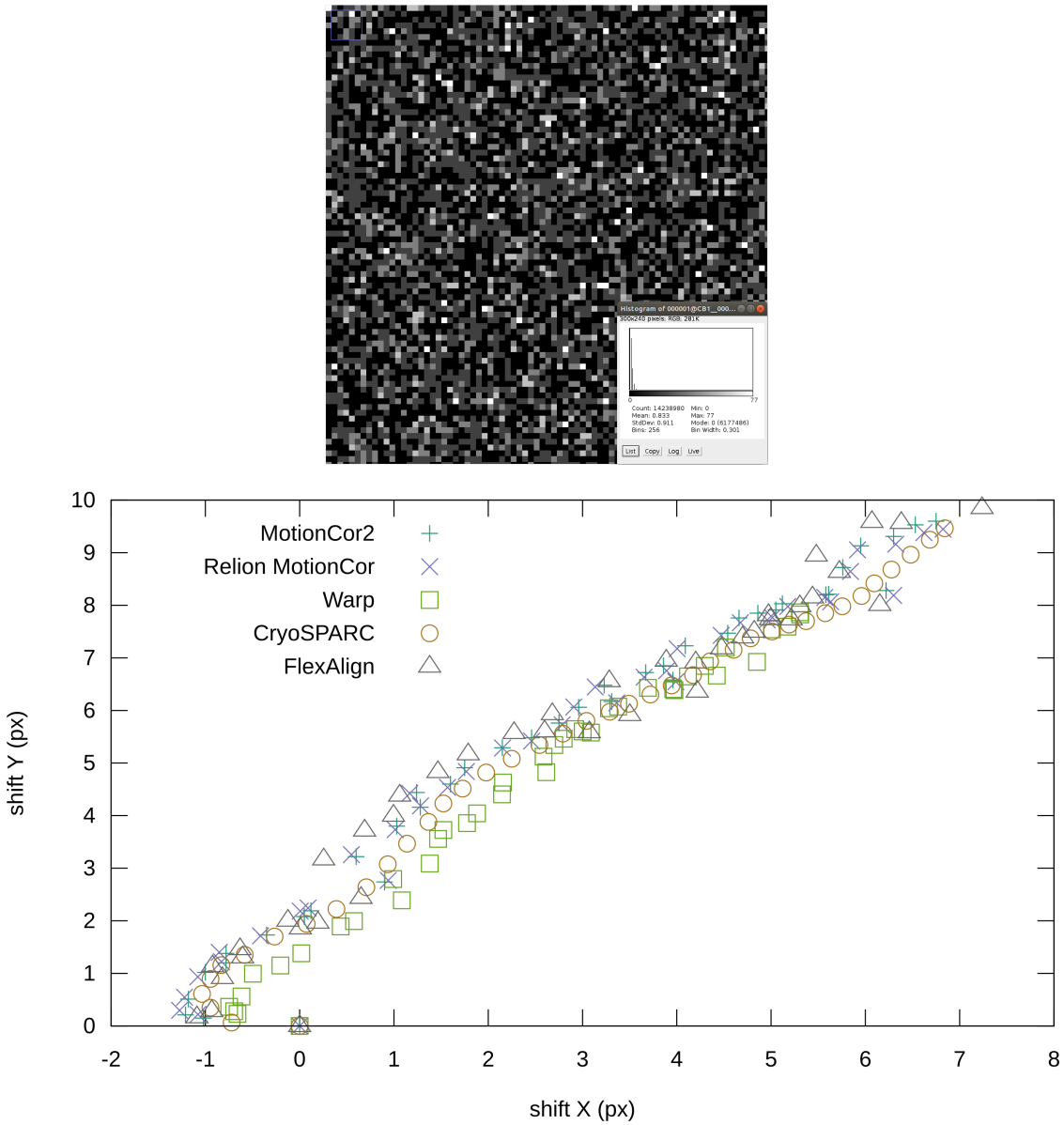

Figure 8. Detail of the frame from the EMPIAR 10288 (normalized) with histogram (before normalization) (**top**), reported global shifts by different programs (**bottom**).

Details of the obtained micrographs are shown in Figure 9. All programs produce relatively sharp micrographs. We can see that MotionCor2 clamps low values a little.

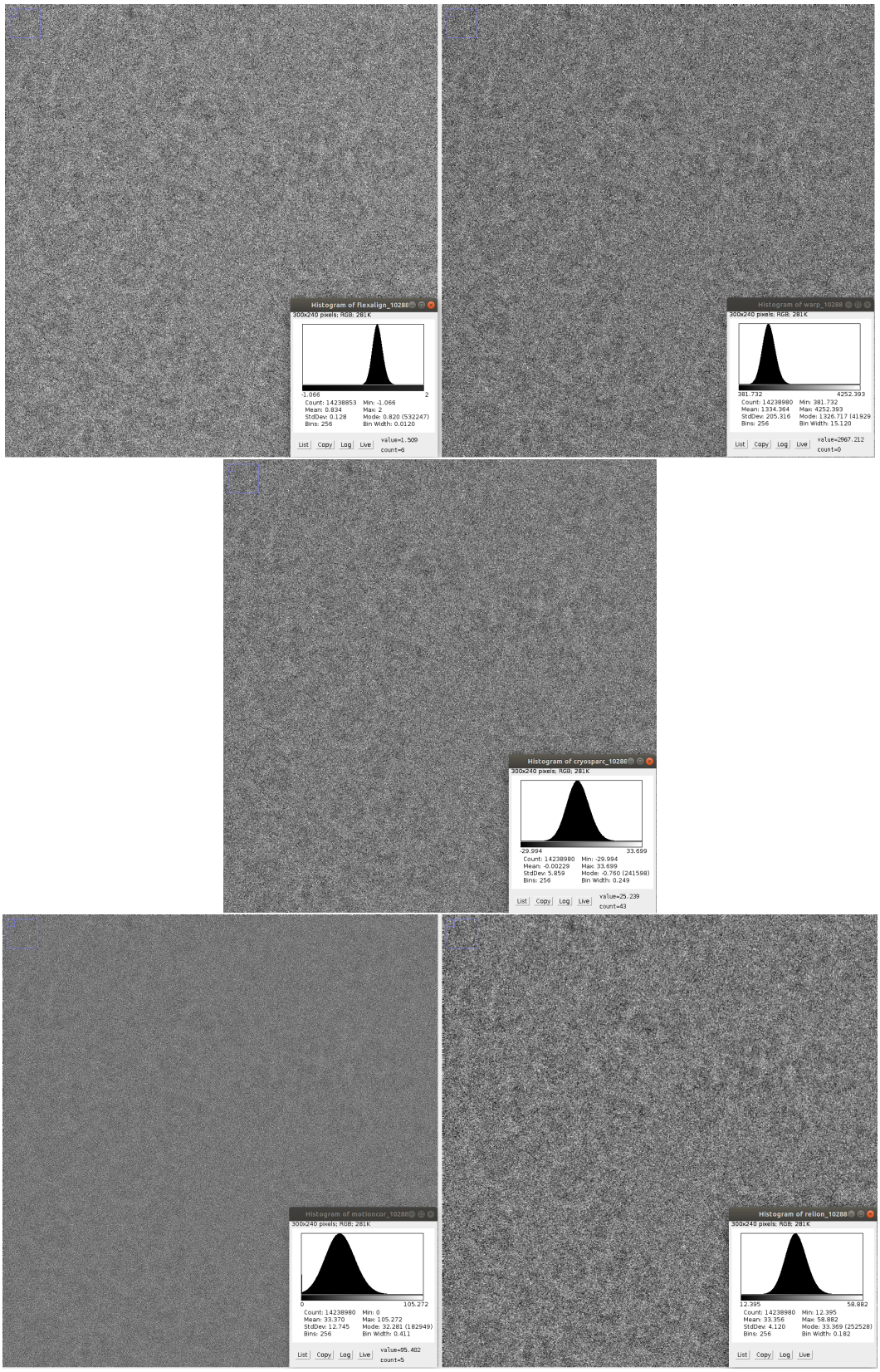

Figure 9. Detail of the produced micrograph using EMPIAR 10288 dataset (normalized) with histogram (before normalization): FlexAlign (**top left**), Warp (**top right**), cryoSPARC (**center**), MotionCor2 (**bottom left**), Relion MotionCor (**bottom right**).

Figure 10 shows the radial average of the Power Spectrum Density (PSD) for the resulting micrographs. As can be seen, micrographs contain useful information up to a resolution between 4 to 5 Å. The power increase at high frequencies of MotionCor2 might indicate some high frequency enhancement, while decreasing tendency of the FlexAlign, Warp, and Relion MotionCor suggest dampening of the high frequencies.

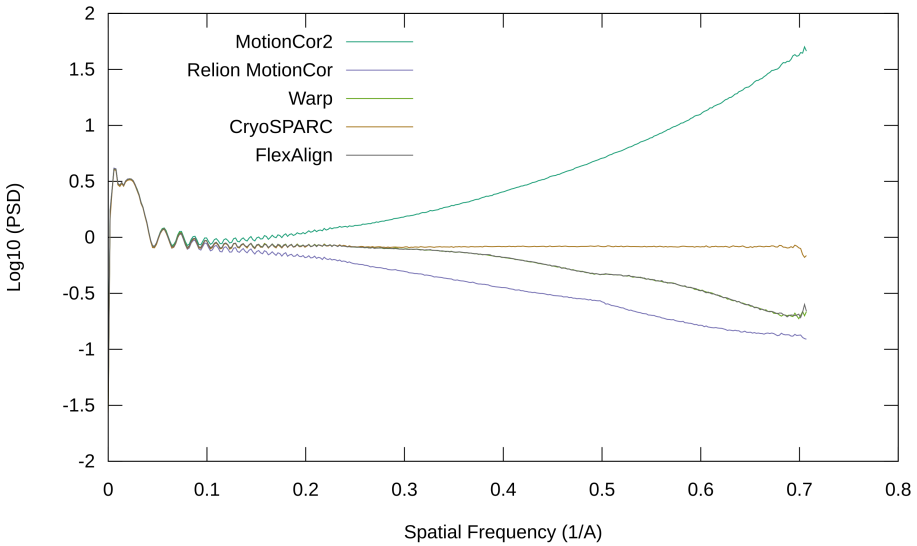

Figure 10. Radial average of the PSD of the produced micrograph using EMPIAR 10288.

4.2.2. EMPIAR 10314

The EMPIAR 10314 dataset consists of movies of 33 frames, each frame having a resolution of 4096 × 4096 pixels and an average exposure of 1.51 $e/\text{Å}^2$. The pixel size is 1.12 Å, so we have used 8 × 8 patches. The camera model is not specified and neither gain nor dark correction images are provided. For Warp, the movie has been converted from original .tif format with AdobeDeflate compression to .mrc, as the former is not yet supported.

Figure 11 shows a trace of the reported global alignment and histogram of the first frame. As we can see from the histogram, movie frames have probably been somehow post-processed. For that reason, FlexAlign in default settings (FlexAlign 30 Å) reports a different shift in the x-direction, compared to other programs. However, the difference is rather small, around 1.2 Å on the span of the whole movie. It is interesting that the total shift in the y-direction is over 8 times higher than in the x-direction. If we change the filtering to 10 Å, FlexAlign (FlexAlign 10 Å) results in a global trajectory that is consistent with other algorithms. This experiment highlights the need for the lowpass filter during the calculation of the relative shifts: it is primarily aimed at making sure that the signal being correlated has enough information (remind that normally more than half of the pixels of the frames have no electron hit, and for a typical dose in an area of 30 Å^2 there are about 450 pixels hit by electrons); if the frame already has enough local information (as in this example), it is best not to blur it.

Details of the obtained micrographs are shown in Figure 12. All programs produce sharp micrographs, without any cropping of the high/low values.

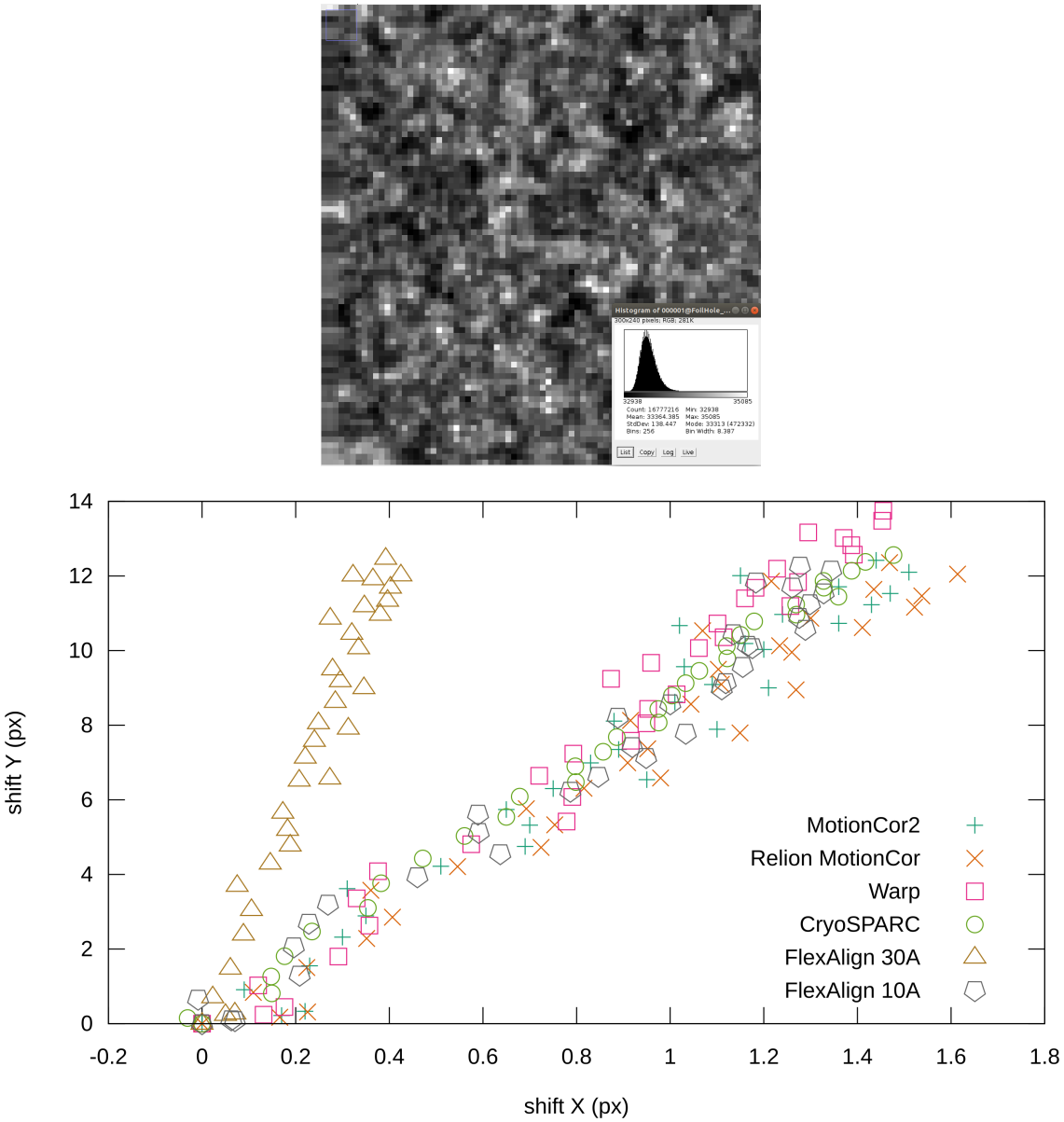

Figure 11. Details of the frame from the EMPIAR 10314 (normalized) with histogram (before normalization) (**top**), reported global shifts by different programs (**bottom**).

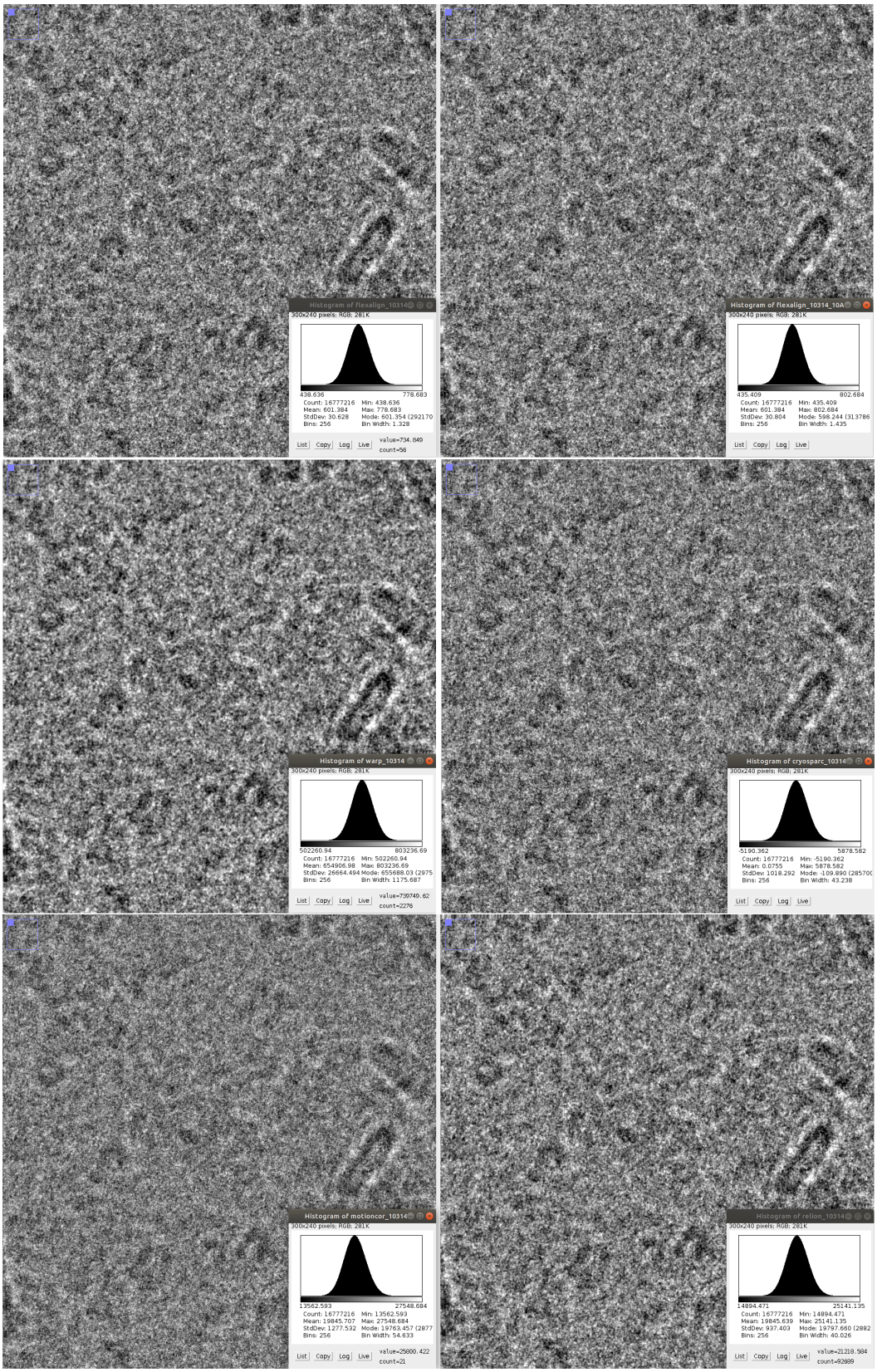

Figure 12. Details of the produced micrograph using EMPIAR 10314 dataset (normalized) with histogram (before normalization): FlexAlign (**top left**), FlexAlign with low-pass filter at 10 Å (**top right**), Warp (**center left**), cryoSPARC (**center right**), MotionCor2 (**bottom left**), Relion MotionCor (**bottom right**).

Figure 13 shows the radial average of the PSD for the resulting micrographs. These micrographs contain frequencies of up to three and a half Å. Again, MotionCor2's PSD might indicate problem with handling high frequencies, while decreasing tendency of the cryoSPARC, FlexAlign, Relion MotionCor and Warp suggests dampening of the high frequencies (either by the movie alignment algorithm, the image interpolation scheme to produce the micrograph, or as originally recorded by the movie). Warp dampens the most, followed by Relion MotionCor, FlexAlign and cryoSPARC. CryoSPARC, FlexAlign, Warp, and Relion MotionCor also seem to preserve bit higher frequencies than MotionCor2.

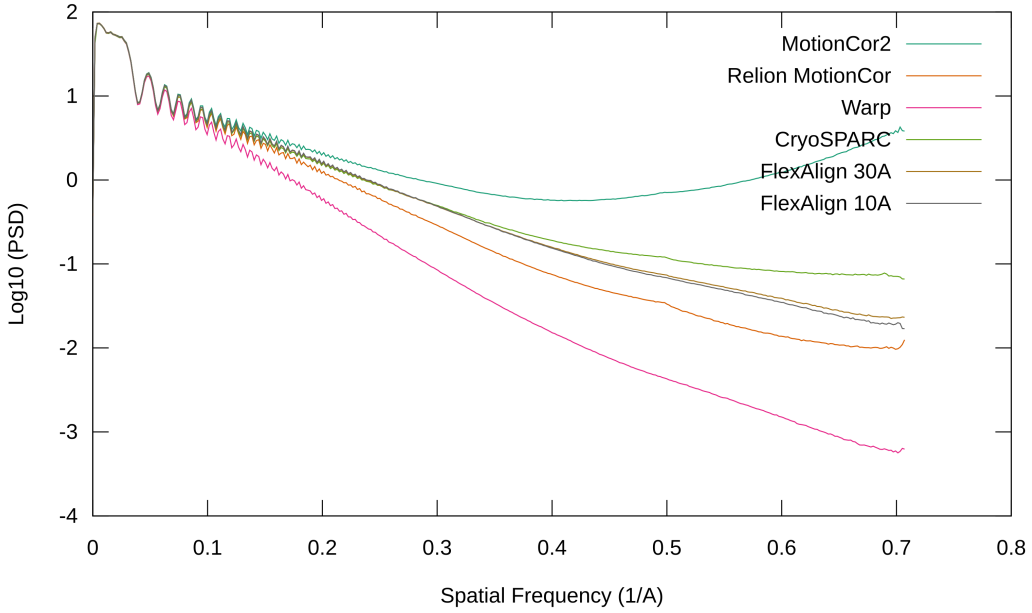

Figure 13. Radial average of the PSD of the produced micrograph using EMPIAR 10314.

4.2.3. EMPIAR 10196

The EMPIAR 101096 dataset consists of movies of 40 frames, each with a size of 7420 × 7676 pixels and an average exposure of 1.264 $e/Å^2$. The pixel size is 0.745 Å (super-resolution) and we used 20 × 21 patches. CryoSPARC used 9 × 10 patches. The camera model is a K2 on a Talos Arctica. Gain is provided along with instructions on application (must be rotated 90 degrees left and flipped horizontally). Figure 14 shows a trace of the reported global alignment and histogram of the first frame. Despite all efforts, we were not able to align this movie with Warp (we were getting zero shift for all frames).

From the reported global shift, we can see that it has an unusually high drift of almost 80 pixels. FlexAlign in default settings checks for shifts of up to 40 pixels (FlexAlign max shift 40) and therefore reports different values as compared to both Relion MotionCor and MotionCor2. This error is partially compensated during the local alignment phase, which is able to compensate for an additional 40-pixel shift. If we allow for higher shifts, FlexAlign (FlexAlign max shift 80) reports similar trace to other algorithms. Details of the obtained micrographs are shown in Figure 15. Again, we can see that MotionCor2 crops the low values. It is worth noticing the difference at the edges of the micrograph—programs handle them differently, and particles coming from these areas should not be used for further steps.

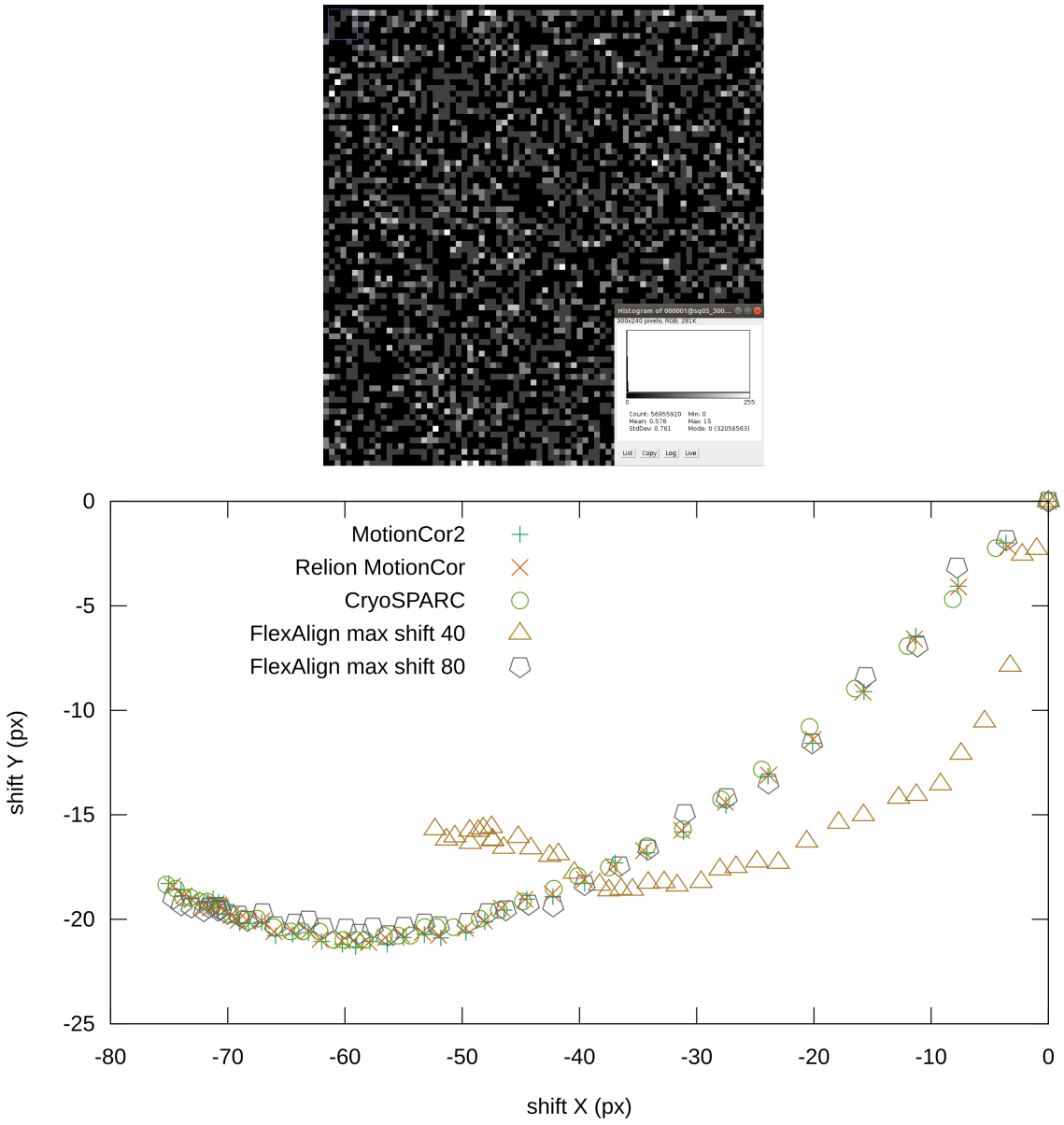

Figure 14. Details of the frame from the EMPIAR 10196 (normalized) with histogram (before normalization) (**top**), reported global shifts by different programs (**bottom**).

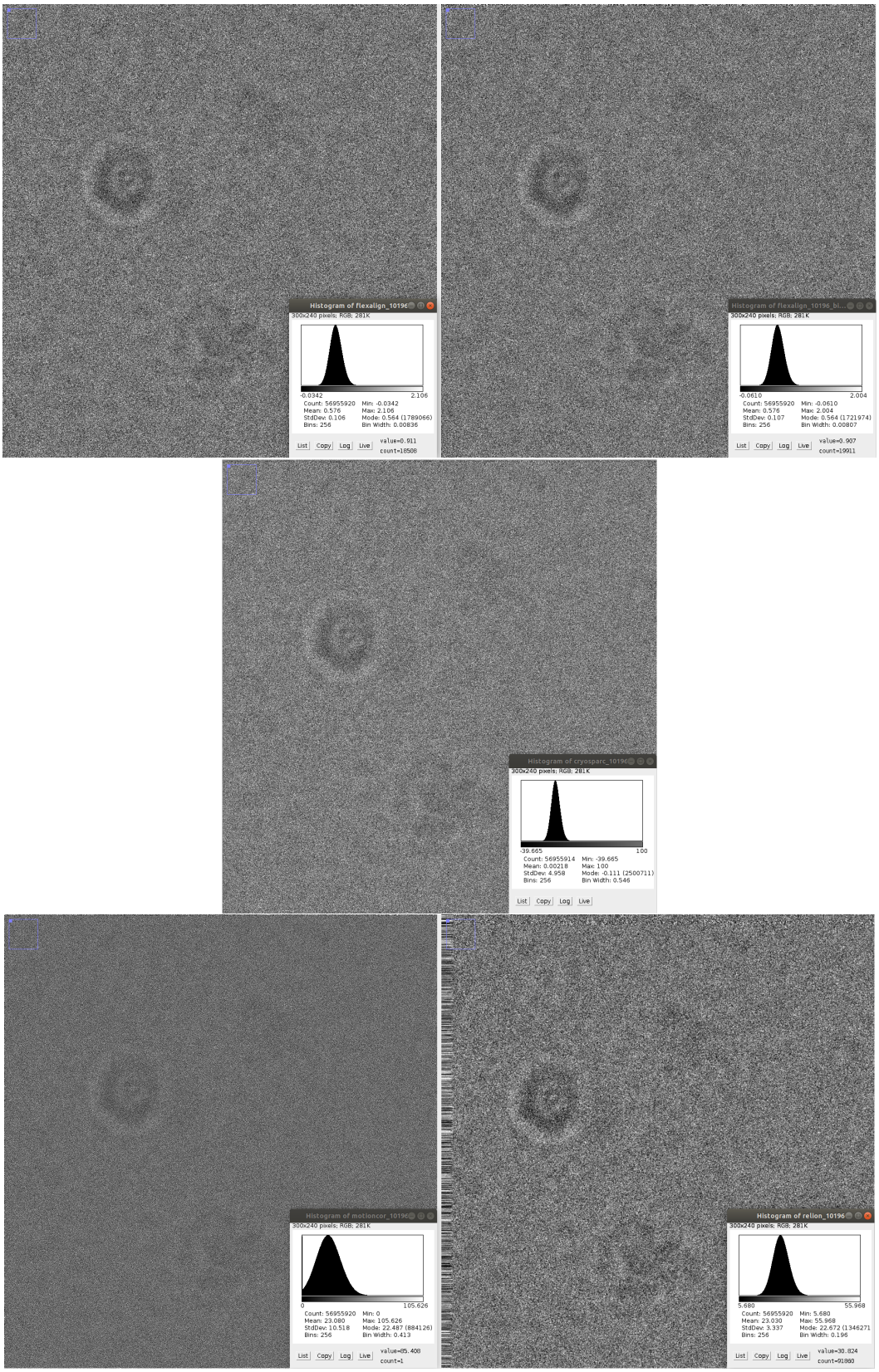

Figure 15. Details of the produced micrograph using EMPIAR 10196 dataset (normalized) with histogram (before normalization): FlexAlign (**top left**), FlexAlign with max shift of 80 px (**top right**), cryoSPARC (**center**), MotionCor2 (**bottom left**), Relion MotionCor (**bottom right**).

Figure 16 shows the radial average of the PSD for the resulting micrographs. These micrographs contain very low frequencies, below 10 Å, and as such, they might be considered for discarding. Again, MotionCor2's PSD has increasing tendency, while FlexAlign and Relion MotionCor show decreasing tendency. Similarly to dataset 10288, cryoSPARC does not seem to increase nor decrease the frequencies.

4.2.4. Number of Patches

In the aforementioned tests, we have automatically set the number of segments/patches that are used by FlexAlign during the local alignment estimation, based on the pixel size and the resolution of the frames. By default, we use patches of 500 × 500 Å that we equally distribute to fully cover the frames, i.e., per dimension we use $n = \lceil R_f / R_p \rceil$ patches, where R_f is resolution of the frame in Å and R_p is resolution of the patch in Å. The first and the last patch are aligned with the edges of the frame, and the rest is equally distributed between them.

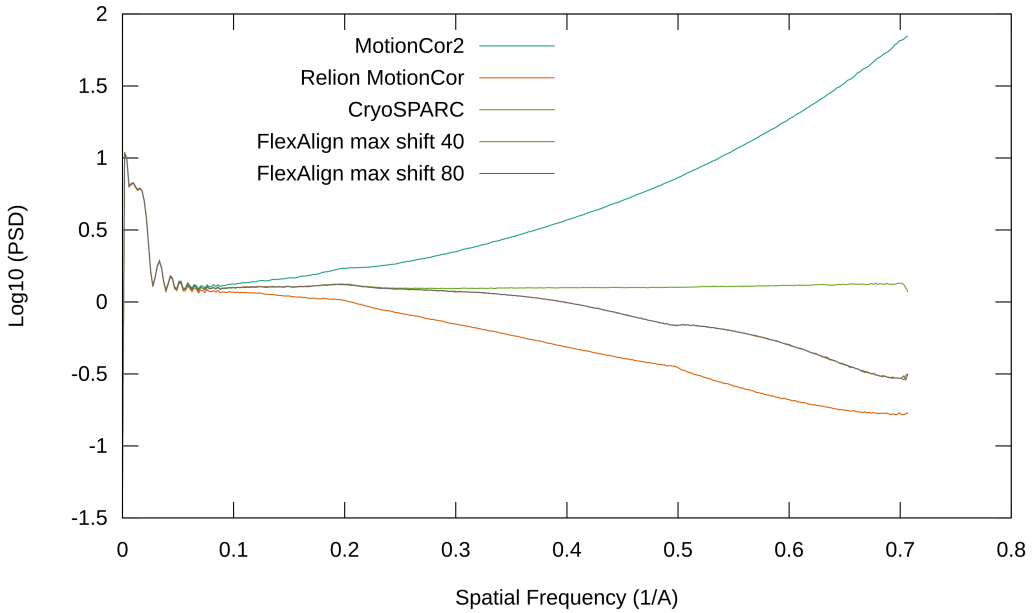

Figure 16. Radial average of the PSD of the produced micrograph using EMPIAR 10196.

Figure 17 shows a 10× magnified estimated total (local + global) shift in the grid of 200 × 100 px, using the Covid-19 spike movie we helped to process. The movie consists of 30 frames of 5760 × 4092 pixels, and pixel size of 1.047 Å per pixel. For this resolution, we would use 12 × 8 patches. As can be seen, even four times fewer patches can give us a general idea of the shift in different parts of the movie. Our default value seems to capture all major deviations. Twice as many patches give us even more detailed insight, but might result in overfitting (see the bottom of the last figure), and as such we believe that it is not necessary during this stage of the processing pipeline. It is also worth noticing that the entire first half of the frames move much more than the second half, while the literature generally mentions a big drift of only the first few frames.

We have also tried a different number of patches for MotionCor2 with our phantom movie. As shown in Figure 18, 5 × 5 patches (The default value when used via Scipion.) are not sufficient to compensate the shift, while 9 × 9 (our default value) result in sharp edges. Neither MotionCor2 nor Relion MotionCor offers an automatic patch size setting, so we used the same values as used by FlexAlign throughout the testing. CryoSPARC does not allow for a manual set of the number of patches.

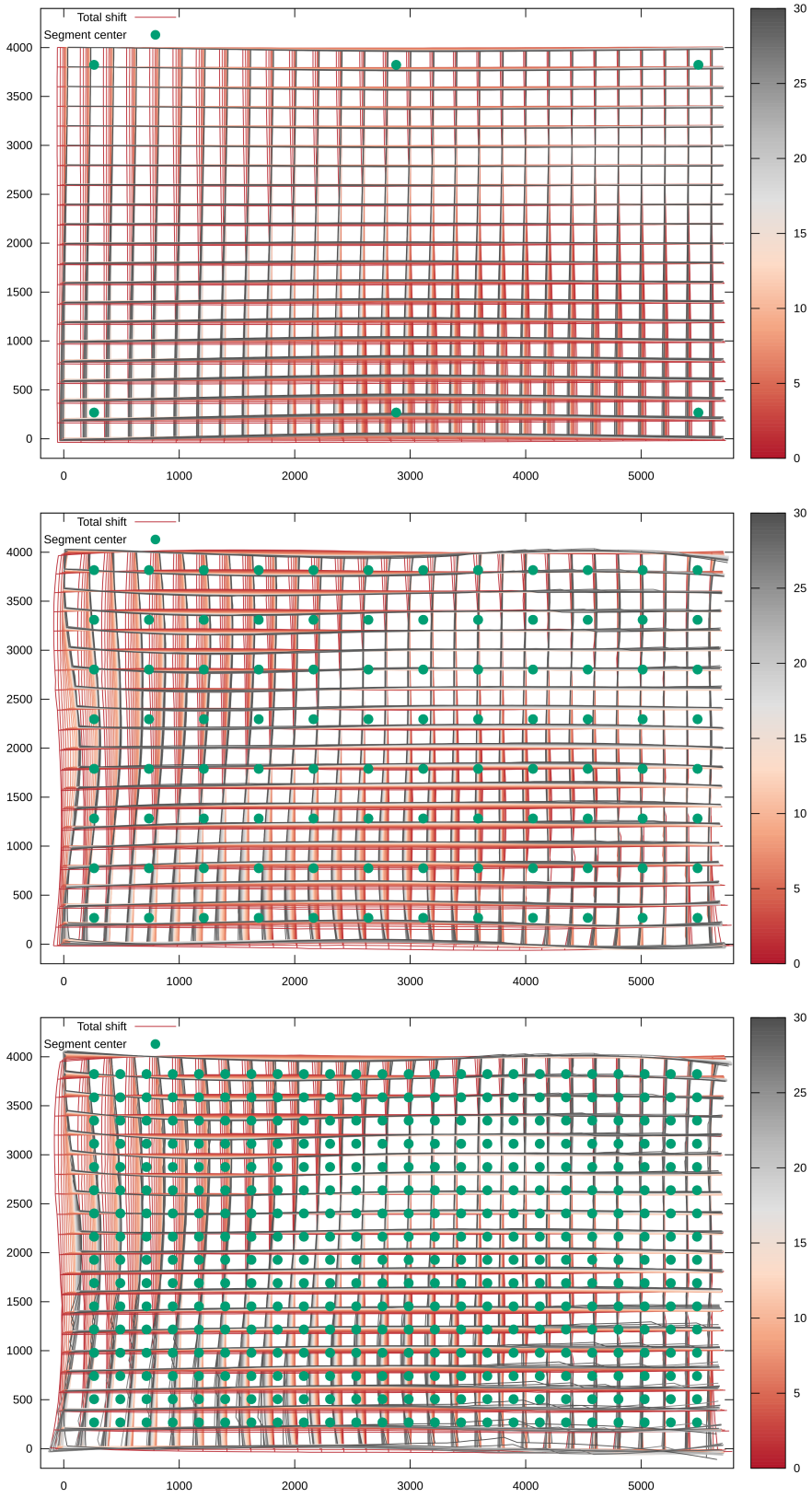

Figure 17. Influence on shift estimation using a different number of patches: 3 × 2 (**top**), 12 × 8 (**middle**, default value), 24 × 16 (**bottom**).

4.3. Performance

We have compared the performance of the FlexAlign, Warp, and MotionCor2 using three machines, as described in Table 2. Relion MotionCor is CPU only and therefore skipped, as well as cryoSPARC, which cannot be set with the same settings. As we could not install MS Windows on the Testbeds 1 and 2, we have compared Warp and FlexAlign using different machines. These results should be taken as illustrative only.

We have tested the most common sizes of the movies, with the default settings for each program, with exception to the number of patches, which was set the same for both programs. The resolution used and the name of the direct detection camera using such a resolution are shown in Table 3. Each configuration has been run five times, and we present the average of these values.

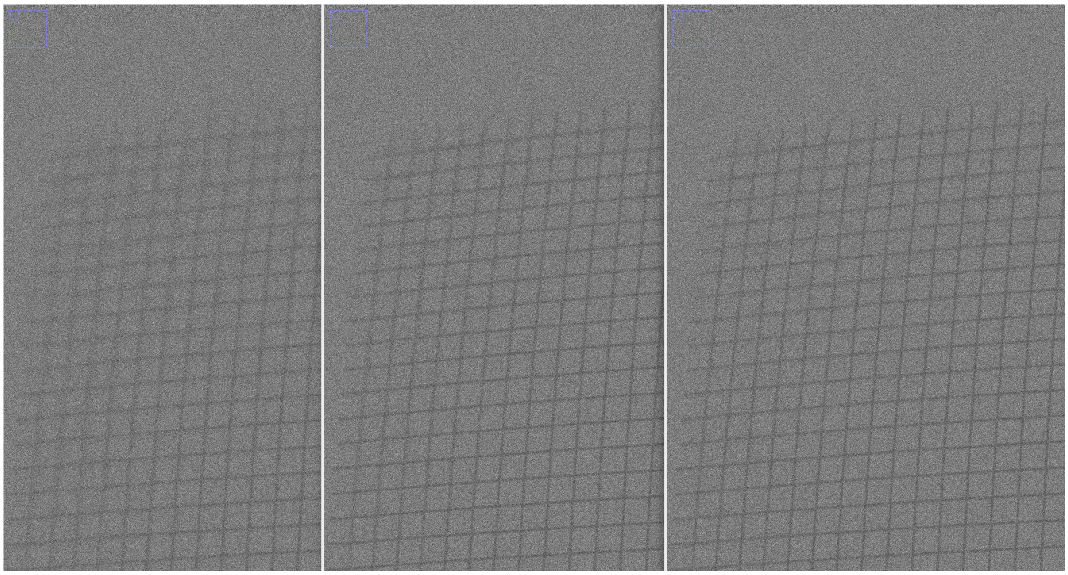

Figure 18. Normalized micrograph of the phantom movie produced by MotionCor2, using different number of patches: 5 × 5 (**left**), 7 × 7 (**middle**), 9 × 9 (**right**).

Table 2. HW used for benchmarking.

	Testbed 1	**Testbed 2**	**Testbed 3**
CPU	Intel(R) Core(TM) i7-8700 (12 cores, 3.20 GHz)		Intel(R) Core(TM) i7-7700HQ (4 cores, 2.80 GHz)
GPU	GeForce RTX 2080	GeForce GTX 1070	GeForce GTX 1060
CUDA/driver	10.1/418.39	10.1/418.67	8.0.61/436.02 (Win 10)/390.116 (Ubuntu 18.04)
SSD	Samsung SSD 970 EVO 500 GB		NVMe TOSHIBA 1024 GB
RAM	2 × 16 GB DDR4 @ 2.6 GHz		2 × 16 GB DDR4 @ 2.4 GHz

Table 3. Resolution and number of frames used for testing.

	Size	**No. of Patches**
Falcon	4096 × 4096 × 40	9 × 9
K2	3838 × 3710 × 40	8 × 8
K2 super (resolution)	7676 × 7420 × 40	16 × 15
K3	5760 × 4092 × 30	12 × 9
K3 super (resolution)	11,520 × 8184 × 20	24 × 17

As discussed in Section 3.4, we dynamically change the size of the patches and frames to speed up the processing. For FlexAlign, we have therefore measured the time without autotuning, with autotuning, and after autotuning, when the new sizes are loaded from the local storage. Since autotuning slows down only the first execution, we also report the minimal number of movies necessary to compensate for this extra time. Results can be found in Table 4 for Testbed 1, Table 5

for Testbed 2, and Table 6 for Testbed 3, where we show Warp and tuned times of FlexAlign only for brevity.

As can be seen, MotionCor2 is very well optimized for all tested sizes and GPU architectures. FlexAlign is on average at 63% of its performance after autotuning (51% without autotuning), but both programs are able to process the movies on-the-fly.

In terms of autotuning, it is more important for less powerful GPU, where it pays-off after as few as 6 movies (13 on average). For high-end GPU, at least 20 movies have to be processed (on average) to compensate for the time spent on autotuning. Since typically hundreds to thousands of movies are processed, we use autotuning by default.

Table 4. Execution time on Testbed 1.

	Falcon	K2	K2 Super	K3	K3 Super
MotionCor2	4.6 s	4.3 s	15.7 s	5.0 s	13.1 s
FlexAlign (tuned)	9.2 s	7.6 s	25.6 s	8.8 s	20.5 s
FlexAlign (autotuning)	49.2 s	34.4 s	71.9 s	59.3 s	72.2 s
FlexAlign (non-tuned)	10.8 s	9.1 s	31.5 s	10.7 s	22.9 s
Movies to pay-off	25	18	8	27	22

Table 5. Execution time on Testbed 2.

	Falcon	K2	K2 Super	K3	K3 Super
MotionCor2	5.5 s	5.2 s	20.3 s	5.9 s	15.7 s
FlexAlign (tuned)	9.0 s	8.2 s	27.3 s	9.3 s	21.1 s
FlexAlign (autotuning)	47.8 s	38.7 s	73.3 s	56.2 s	65.7 s
FlexAlign (non-tuned)	11.7 s	10.4 s	35.2 s	11.5 s	26.9 s
Movies to pay-off	15	14	6	22	8

Table 6. Execution time on Testbed 3.

	Falcon	K2	K2 Super	K3	K3 Super
Warp	11.7 s	10 s	14.2 s	13.1 s	15.6 s
FlexAlign (tuned)	11.9 s	9.9 s	35.5 s	11.3 s	27.7 s

5. Conclusions

In this paper, we have presented our new program for movie alignment, called FlexAlign. FlexAlign is a GPU accelerated program able to correct both the global and local shifts of the movies. Using current generations of GPUs, our program is able to process the most common sizes of the movies on-the-fly, though it is slower than its direct competitor, MotionCor2. Compared to MotionCor2, FlexAlign produces micrographs with higher contrast, and it seems to be more resilient to noise than both MotionCor2 and Relion MotionCor. It also tends to preserve higher frequencies in the resulting micrograph. Last but not least, FlexAlign stores the data necessary to track the particle movement with each micrograph to allow for precise particle polishing, in a compact way of the B-spline coefficients.

In future releases, we plan to address several bottlenecks that we have identified in our current implementation, and also to use the information on the local shift during the particle picking and polishing.

FlexAlign is implemented in Xmipp [17], an open-source suite of Cryo-EM algorithms available under GNU General Public License. As part of Xmipp, it is also freely available in the Scipion [18] framework.

Author Contributions: Methodology: J.F. and C.Ó.S.S.; Software: D.S. and A.J.-M.; Text: D.S.; Supervision: J.M.C. All authors have read and agreed to the published version of the manuscript.

Funding: The authors would like to acknowledge financial support from: the Comunidad de Madrid through grant CAM (S2017/BMD-3817), the Spanish Ministry of Economy and Competitiveness (BIO2016-76400-R), the Instituto de Salud Carlos III, PT17/0009/0010 (ISCIII-SGEFI/ERDF) and the European Union and Horizon 2020 through grant: CORBEL (INFRADEV-01-2014-1, Proposal 654248), INSTRUCT-ULTRA (INFRADEV-03-016-2017, Proposal 731005), EOSC Life (INFRAEOSC-04-2018, Proposal: 824087) and HighResCells (ERC-2018-SyG, Proposal: 810057). The project that gave rise to these results received the support of a fellowship from "la Caixa" Foundation (ID 100010434). The fellowship code is LCF/BQ/DI18/11660021. This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 713673. The work was supported from European Regional Development Fund-Project "CERIT Scientific Cloud" (No. CZ.02.1.01/0.0/0.0/16_013/0001802).

Acknowledgments: The authors acknowledge the support and the use of resources of Instruct, a Landmark ESFRI project.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Glaeser, R. *Methods in Enzymology*; Academic Press: Cambridge, MA, USA, 2016; *579*, pp. 19–50, doi:10.1016/bs.mie.2016.04.010.
2. Hattne, J.; Martynowycz, M.W.; Penczek, P.A.; Gonen, T. MicroED with the Falcon III direct electron detector. *IUCrJ* **2019**, *6*, 921–926, doi:10.1107/S2052252519010583.
3. Borgnia, M.J.; Bartesaghi, A. Practices in Data Management to Significantly Reduce Costs in Cryo-EM. *Microsc. Microanal.* **2019**, *25*, 1378–1379.
4. Danev, R.; Yanagisawa, H.; Kikkawa, M. Cryo-Electron Microscopy Methodology: Current Aspects and Future Directions. *Trends Biochem. Sci.* **2019**, *44*, 837–848, doi:10.1016/j.tibs.2019.04.008.
5. Marko, A. CryoEM Takes Center Stage: How Compute, Storage, and Networking Needs are Growing with CryoEM Research. 2019. Available online: https://www.microway.com/hpc-tech-tips/cryoem-takes-center-stage-how-compute-storage-networking-needs-growing (accessed on 22 June 2020)
6. Zheng, S.Q.; Palovcak, E.; Armache, J.P.; Verba, K.A.; Cheng, Y.; Agard, D.A. MotionCor2: Anisotropic correction of beam-induced motion for improved cryo-electron microscopy. *Nat. Methods* **2017**, *14*, 331–332.
7. Tegunov, D.; Cramer, P. Real-time cryo-electron microscopy data preprocessing with Warp. *Nat. Methods* **2019**, *16*, 1146–1152, doi:10.1038/s41592-019-0580-y.
8. Abrishami, V.; Vargas, J.; Li, X.; Cheng, Y.; Marabini, R.; Sorzano, C.Ó.S.; Carazo, J.M. Alignment of direct detection device micrographs using a robust optical flow approach. *J. Struct. Biol.* **2015**, *189*, 163–176.
9. Zivanov, J.; Nakane, T.; Forsberg, B.O.; Kimanius, D.; Hagen, W.J.; Lindahl, E.; Scheres, S.H. New tools for automated high-resolution cryo-EM structure determination in RELION-3. *Elife.* **2018**, *7*, e42166.
10. Jonic, S.; Sanchez Sorzano, C.O. *Optical and Digital Image Processing: Fundamentals and Applications*; John Wiley & Sons, Ltd.: Hoboken, NJ, USA, 2011; Chapter 6, pp. 119–134, doi:10.1002/9783527635245.ch6.
11. NVIDIA. *CUFFT Library User's Guide*; NVIDIA: Santa Clara, CA, USA, 2019.
12. Střelák, D.; Filipovič, J. Performance Analysis and Autotuning Setup of the CuFFT Library. In Proceedings of the 2nd Workshop on AutotuniNg and ADaptivity AppRoaches for Energy Efficient HPC Systems, Limassol, Cyprus, 4 November 2018; doi:10.1145/3295816.3295817.
13. Punjani, A.; Rubinstein, J.L.; Fleet, D.J.; Brubaker, M.A. cryoSPARC: algorithms for rapid unsupervised cryo-EM structure determination. *Nat. Methods* **2017**, *14*, 290–296. doi:10.1038/nmeth.4169.
14. Krishna, K.K.; Shalev-Benami, M.; Robertson, M.; Hu, H.; Banister, S.; Hollingsworth, S.; Latorraca, N.; Kato, H.; Hilger, D.; Maeda, S.; et al. *Cryo Electron Microscopy of Cannabinoid Receptor 1-G Protein Complex*; EMBL-EBI: Cambridgeshire, UK, 2019; doi:10.6019/EMPIAR-10288.
15. Suga, M.; Ozawa, S.; Yoshida-Motomura, K.; Akita, F.; Miyazaki, N.; Takahashi, Y. Structure of the green algal photosystem I supercomplex with a decameric light-harvesting complex I. *Nat. Plants* **2019**, *5*, 626–636, doi:10.1038/s41477-019-0438-4.
16. Nureki, O.; Kasuya, G.; Nakane, T.; Yokoyama, T.; Jia, Y.; Inoue, M.; Watanabe, K.; Nakamura, R.; Nishizawa, T.; Kusakizako, T.; et al. Cryo-EM structures of the human volume-regulated anion channel LRRC8. *Nat. Struct. Mol. Biol.* **2018**, *25*, 797–804. 2018, doi:10.1038/s41594-018-0109-6.

17. De la Rosa-Trevín, J.; Otón, J.; Marabini, R.; Zaldivar, A.; Vargas, J.; Carazo, J.; Sorzano, C. Xmipp 3.0: An improved software suite for image processing in electron microscopy. *J. Struct. Biol.* **2013**, *184*, 321–328.
18. De la Rosa-Trevín, J.; Quintana, A.; Del Cano, L.; Zaldivar, A.; Foche, I.; Gutiérrez, J.; Gómez-Blanco, J.; Burguet-Castell, J.; Cuenca-Alba, J.; Abrishami, V.; et al. Scipion: A software framework toward integration, reproducibility and validation in 3D electron microscopy. *J. Struct. Biol.* **2016**, *195*, 93–99.