Swipe Mosaics from Video - Supplemental Material

Malcolm Reynolds, Tom S. F. Haines and Gabriel J. Brostow

1 Baseline Algorithms

In this document we provide further results of comparisons between our system and Wu *et al*.'s VisualSFM [1], [2], Viewfinder Alignment (VfA) of Adams *et al*. [3], a "micro-SfM" method which computes a 2D translation using SIFT matching with RANSAC, and direct (all-pixel) methods as summarised by Szeliski [4]. For each baseline algorithm we have the capability to load the output into our viewer for qualitative comparison. To load VisualSfM's output of 6d.o.f. camera positions into the Swipe Mosaic viewer we project each camera to a location & orientation on the 2D plane. We do this by computing a normal vector from the average "forward" direction over all cameras, then projecting each camera perpendicularly onto a plane defined by this normal. The location on this plane gives a 2D coordinate to be loaded into the viewer (see main paper).

1.1 Direct Methods

An excellent summary of techniques for Image Alignment and Stitching was presented by Szeliski [4]. Chapter 3, "Direct (pixel-based) alignment", details a number of methods to compute a 2D alignment between image pairs. The general approach is to define some error metric which can evaluate how well each potential 2D alignment matches the contents of each image. Given the error metric, one can exhaustively evaluate all possible alignments, or use a coarse to fine method to limit the amount of computation, and the alignment which produces the lowest error is chosen. Various error metrics can be defined on overlapping pixels, such as Sum of Square Differences or Sum of Absolute Differences. We compare to the popular method of Normalized Cross Correlation (NCC), which we also used as the basis of our feature vector computation. NCC is an improvement over improves over Cross Correlation (which has a tendency to give incorrect offets in the presence of large high intensity areas) by normalising the overall intensity of each of the regions being compared. However, as noted by Szeliski, *"[NCC's] performance degrades for noisy low-contrast regions"* so the improved technique is not immune to problems.

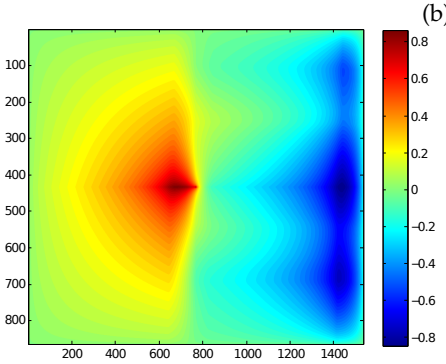
Two temporally adjacent frames (Fig. 1a and Fig. 1b) from the PRISM sequence were selected. Some scene geometry is visible on the left of the image, but most of the pixels have been overloaded by the bright light and are reporting close to perfect white. Nevertheless, it is clear that horizontal camera motion has taken place, given the parts of the geometry which we *can* see. The NCC image computed in MATLAB is shown in Fig. 1c. Two closeups of the region around the peak are shown in Fig. 1d and Fig. 1e. Note that the right hand peak in Fig. 1d has a higher NCC value, but the location $(768, 432)$ implies that *zero* translation is the optimal image alignment. The left hand peak in Fig. 1e has a slightly lower magnitude, but the offset $(679, 432)$ indicates a horizontal offset of $(768 - 679)/2 = 39$. Indeed, a translational shift of $(39, 0)$ does bring the two images into good alignment. Our RRF based system produced an estimate with mean $(0.0169, 0.0004)$ and variance $(0.00132, 0.00168)$ for these images (note these results are not in units of pixels as above, so cannot be directly compared), showing primarily horizontal motion with roughly isotropic variance, as we would expect from the fact that the visible texture in the image confirms no vertical motion has taken place. Any algorithm which simply computes NCC over entire images, finds the single peak and uses the value will be prone to fail in image pairs such as this, whereas our system produces a translation in the correct direction. Note that un-normalized Cross Correlation for this image actually produced a *purely vertical* translation, reinforcing the idea that un-normalized cross correlation is unsuitable for large high intensity regions, performing even worse than NCC.

Our method returns a distribution over possible camera motion, which is a great advantage in cases of ambiguity such as repeated structure. Fig. 2 shows how in the presence of multiple potential alignment Another example of how our method is superior to NCC is in shown in Fig. 2. When repeated structure creates a number of possible alignments, our method (Fig. 2d) returns an anisotropic estimate compared to the deterministic estimate provided by NCC (Fig. 2c). Corresponding repeated structure is shown in the NCC response image; the peak happens to be located in a

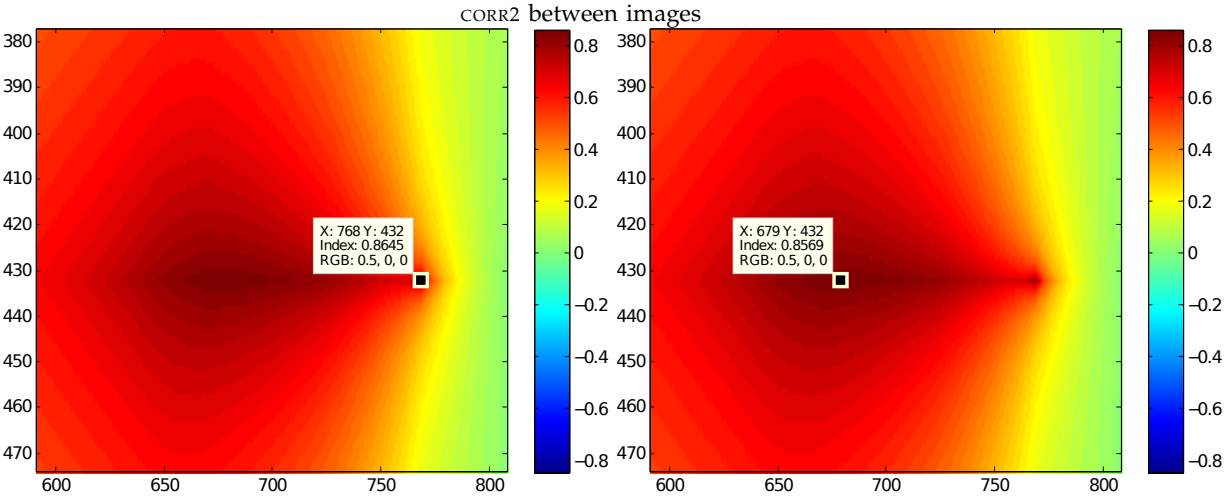
• *The authors are with University College London.*

(a) PRISM frame 165

(b) PRISM frame 166



(c) NCC result from MATLAB NORMXCORR2 between images



(d) Primary peak (greatest NCC value)

(e) Secondary peak (lower NCC value)

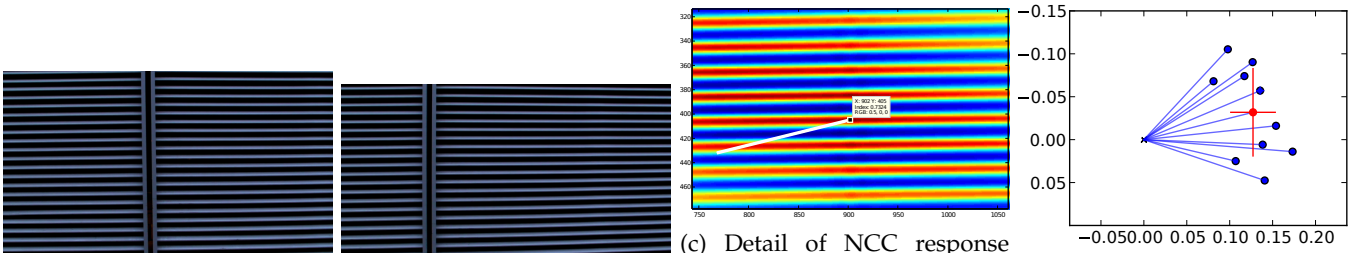
Fig. 1: NCC failing to compute the correct offset for two frames in PRISM. Black borders added to top images for clarity.

ridge which indicates slight upward motion (as well as rightwards). The vertically adjacent ridges have similar NCC values and (given the magnitude of the camera motion between the images) are surely almost as likely. However the single transformation returned by pure NCC alignment will not represent this information at all. The result from our method is more desirable in this situation.

1.2 Structure from Motion: VisualSfM

We selected the SCULPTURE (Fig. 3a) and LEAVES (Fig. 3b) sequences as likely to cause SfM failure. SCULPTURE includes specularities, motion blur, and has few suitable corners for interest point detection. LEAVES contains lots of geometry suitable for interest point detection, but most of these areas are on leaves, which are being blown around in the wind, meaning points detected on them may adversely contribute to the optimisation.

VisualSfM was run on each sequence in both ordered and unordered mode. This mode affects which image



(a) Image A (b) Image B (c) Detail of NCC response with peak and zero location marked. (d) Distribution returned by our method.

Fig. 2: NCC computes a large diagonal offset for 2 frames containing repeated structure. The white line in c) connects the NCC peak with the location which would represent zero offset

pairs are compared to find interest points; either all pairs (unordered) or only temporally adjacent pairs (ordered). Running unordered on SCULPTURE, 38 camera locations were reconstructed from the 101 input images, leading to an incomplete Swipe Mosaic. Running ordered mode yields an even worse result, reconstructing 35 camera locations but in three independent groups. The locations that VisualSfM did produce were accurate, but the full camera path produced by our system is preferable, as shown in Fig. 4.

(a) SCULPTURE (b) LEAVES

Fig. 3: Screenshots in the Swipe Mosaic interface of the datasets on which we compare our performance to SfM. The minimap in the bottom right shows the camera locations.

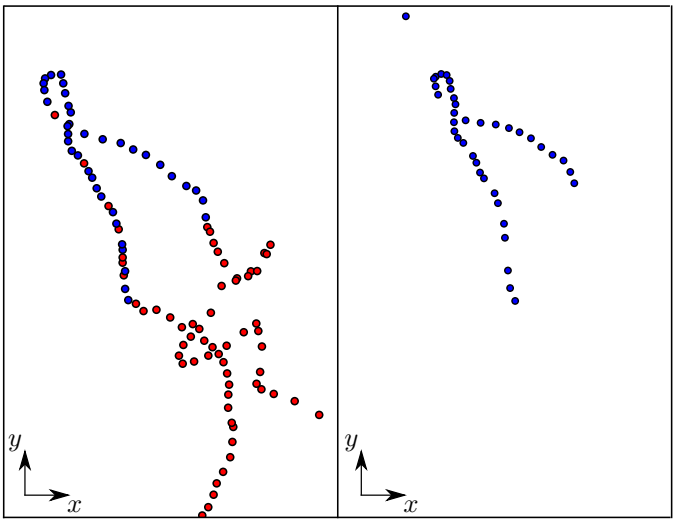


Fig. 4: 2D camera coordinates (unitless) for SCULPTURE produced by our system (left) and VisualSFM (right). Blue points indicate images where both systems gave an estimate of location (note the estimates differ); red points are images where only our system produced an estimate. Note the obvious outlier at the top of the SfM result.

The LEAVES dataset consist of 201 images. VisualSfM in ordered mode computed locations for only 70 of the images, albeit producing a reasonable Swipe Mosaic. Running SfM in unordered mode computes a location for all the images, but the placement undergoes a catastrophic failure, with the resulting Swipe Mosaic suffering severe artifacts. The failure takes the form of one side of the horizontal path being relatively correct, and the image locations gradually worsening as we travel along the video timeline, until the predicted image locations do not even overlap (see supplemental video). For both the sequences in this section, our system produced easily navigable locations for all cameras (see video and Fig. 3). We surmise that unordered mode producing better results in this case was due to a greater variety of image pairs being run through SIFT matching, rather than merely a few temporal neighbors. If a several consecutive frames of video are blurred or contain confusing motion, unordered mode will still be able to search for SIFT matches between frames "either side" of the problem area, thus providing a more robust solution.

1.3 μSfM

"micro-SfM" or "μSfM" is a system which we have developed with the intention of it being an equivalent system to SfM, but without computing any structure, and with camera transforms limited to 2D translation. The thinking behind this is that computing a 6D quantity for each frame is an inherently harder task than computing a 2D quantity for each frame, and so simply comparing our 2D RRF method to VisualSfM was not a fair comparison. Rather, we should apply the technique from VisualSfM to the strictly easier problem of computing translations (not fundamental matrices) in order to compare like-for-like results. "μSfM" matches two images by generating SIFT descriptors and performing matching using the standard algorithm of Lowe [5]. A translation is computed using RANSAC to iteratively select a random SIFT match, compute the corresponding 2D transform and count the number of inliers. The transform with the highest inlier count is used to generate a final refined transform from the entire inlier set. To combine multiple translation estimates across an

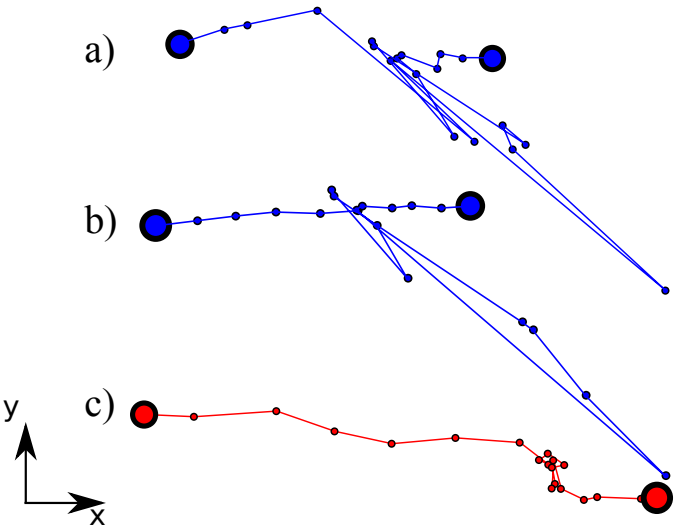


Fig. 5: Regularized locations for the VINYL scene, for the images which straddle the obstruction. Dots represent image locations and the line joins them in temporal order. Larger dots show the start and end of this subsequence. As the correct camera motion is approximately a constant horizontal velocity, the ideal result would be equally spaced dots on a horizontal line. Scale between the 3 diagrams is not meaningful. a): μSfM result with unweighted regularization. b): μSfM with weighted regularization. c): our system.

image sequence, we use the least-squares based layout algorithm developed for our RRF estimates. As the layout algorithm allows for a weighting to be applied to each relative transform (in our main system we use the inverse variance from the forest) we test two versions of μSfM - one unweighted, and one weighted using the ratio of inliers to total number of matches found when generating the transform. This should ensure that translations for which every single SIFT match agrees are given more weight by the optimisation.

Considering the simplicity of the method, μSfM is surprisingly capable. In particularly, it can produce just as good a camera path for the LEAVES sequence as our technique. However as the method relies entirely on interest point matching, we know it is susceptible to fail in the presence textureless regions, motion blur or repeated structure. The VINYL sequence contains a blurry obstruction which is very close to the camera, separating to regions containing strong texture information. The camera travels horizontally, starting in one textured region, passing the obstruction (which takes up the whole screen for a few frames) and ends viewing the second textured region. Surprisingly, inside the (apparently) textureless region, SIFT *is* able to detect a few interest points. Matching these interest points prooves difficult however; most of them have extremely similar appearances, and despite implementing Lowe's technique for avoiding ambiguous matches, the translations returned from the middle frames in this sequence were extremely noisy. Note the results in Fig. 5, bearing in mind the ideal answer would be almost pure horizontal motion. Both μSfM results display problems with some frames ending up at a large displacement to the lower right corner of the map, with the subsequent frames on the normal timeline. It can be seen that the weighted version displays a smoother timeline at the beginning and end of the sequence, but for both a) and b) the mistakes in the middle of the sequence make this difficult to navigate in our interface (frames displaying the obstruction are incorrectly displayed amongst the frames of texture objects). By contrast our result, c), whilst by no means perfect, is a vast improvement on both μSfM results. For the images where texture is available it computes a consistent horizontal motion. For the frames containing no texture, there is insufficient information to state which (if any) direction the camera has moved, so our system returns a number of zero mean, wide variance offsets. The optimisation places these roughly on top of each other, generating the point cluster in the result. Obviously this is not actually correct, as the camera was always moving, but as there is no way to tell this simply from the images pairs we produce a reasonable result, which allows the sequence to be browsed as a Swipe Mosaic without artifacts. It may be possible to improve this aspect of the system by using a camera motion model in the layout algorithm, meaning that when we knew the first few frames had the camera move to the right, then when presented with insufficient visual information our estimate would be some kind of rightwards motion, rather than zero mean motino. Another failure case for μSfM was PRISM, where the camera autogain causes whiteout for a few frames. Similarly to the previous sequence, no reliable feature matches could be detected during this central part of the image sequence, leading to incorrect matches in the middle of the sequence again.

1.4 Viewfinder Alignment

Our final baseline comparison is to Viewfinder Alignment (VfA) of Adams *et al*. [3] . VfA is a method to compute constrained transforms between temporally close video frames. VfA computes a "digest" for each frame by encoding edge information at 4 equally spaced orientations using gradient integral projection arrays, as well as detecting the top k peaks in the image. Two image digests are aligned by first calculating a single 2D shift which best aligns the edge information stored for each image. This 2D shift is applied to the detected corners of one of the digests. The number of inliers (a pair of points, one from each image, landing within 3 pixels from each other) between the two points sets is counted and taken as the confidence that the images have been aligned correctly. The set of inliers is used to generate a similarity transform, giving 4 degrees of freedom (translation, rotation and scale) between pairs of frames. Note that our VfA test scenes were chosen intentionally so that the rotation and scale change was negligible so these parameters are ignored, *i.e*. we are only interested in the relative accuracy of the *translations* computed by different methods.. Experiments showed

that when compute the scale change was typically between 0.98 and 1.02, and the rotation on the order of 0.1 radians, justifying this decision.

VfA has a number of attractive properties, including computational efficiency and being extremely resistant to noise. A disadvantage of the algorithm is that it is completely deterministic, in that only one 2D translation between each frame pair is considered, when the digest edge information could be used to produce a distribution of translations. Additionally, the corners returned from the corner detector are simply stored as 2D locations, without any kind of descriptor, allowing corners which represent different scenes points to potentially be aligned with each other and treated as an inlier. We compared our system to our own re-implementation of VfA. This code is supplied as supplemental material. We now present detailed analysis of VfA on various test scenes.

All the sequences in this document were run through our re-implementation of Viewfinder Alignment. We tried to match each digest with the digests from other images which were within 6 frames (forward or backwards). If a complete graph could be constructed, we used all the inferred translation values as input to our linear least squares regularization (see main paper). Inlier matrices are shown using the standard Jet colormap, except that pairs which either produced zero inliers or were not compared (*i.e.* they were too far apart temporally) are left blank.

Successes

Lobby

Fig. 6: Sample Images from `lobby` sequence

The lobby sequence is largely featureless, but the objects that are seen display strong vertical and horizontal edges, resulting in an excellent output when browsed as a Swipe Mosaic.

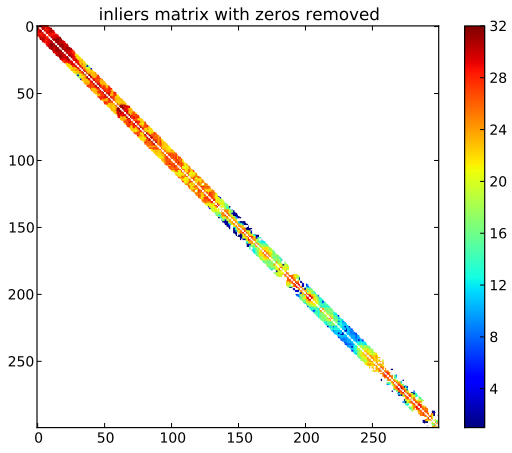


Fig. 7: number of inliers from lobby sequence

Fence

Fig. 8: Sample Images from `fence` sequence

The fence contains repeated structure with many similar looking horizontal edges but VfA is robust to this, producing a fully connected set of inliers (Fig. 9) and a correct reconstruction.

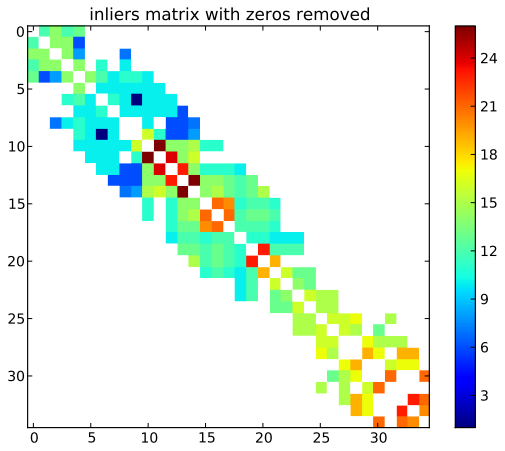


Fig. 9: number of inliers from fence sequence

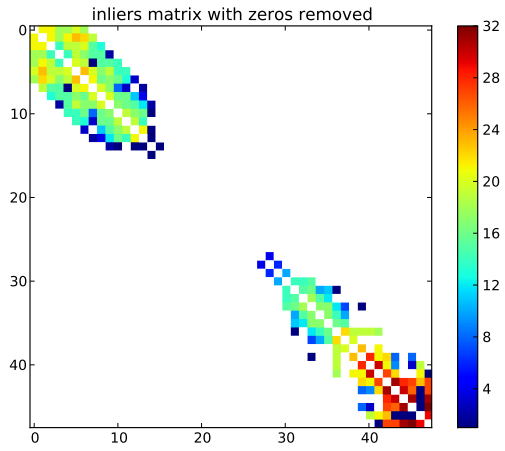


Fig. 11: number of inliers from vinyl sequence. Note two separate "islands"

Failure cases

Vinyl

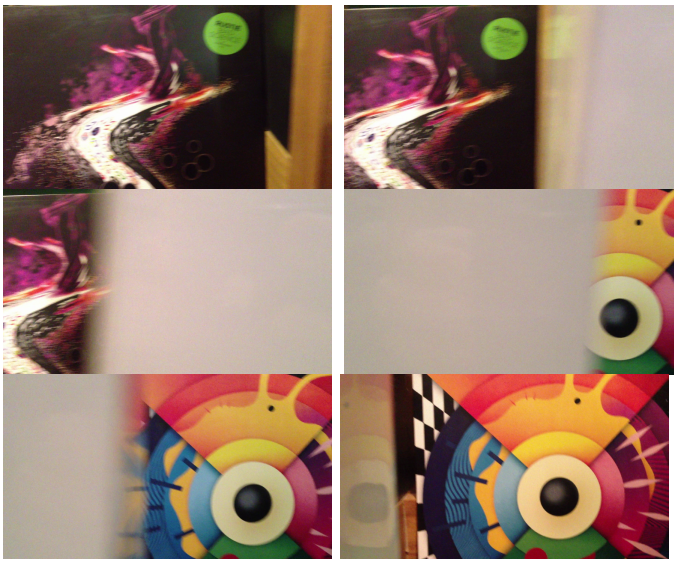


Fig. 10: Sample images from `vinyl` sequence

The Vinyl sequence contains an obstruction which does not trigger Viewfinder Alignment's corner detection (Fig. 10). This causes a large region with zero inliers (Fig. 11) which prevents the start and end of this sequence from connecting to each other, and therefore renders it impossible to create a single set of camera paths to be loaded into our viewer.

Grating

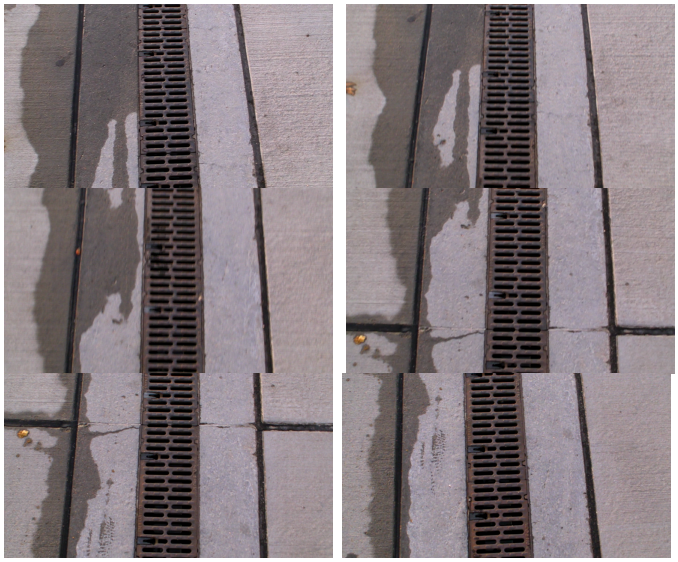


Fig. 12: Sample images from `grating` sequence

The Grating sequence is an interesting case because it contains very easily localisable vertical edges but relatively few unambiguous horizontal edges (Fig. 12). Viewfinder Alignment finds sufficient transforms to regularise the 80 frame segment all together as one connected cluster, leading a promising looking inliers graph (Fig. 13). However, a small number of incorrect matches corrupt the whole regularisation, resulting in final camera locations shown in Fig. 14 and Fig. 15. The correct arrangement should be a roughly straight vertical line. It seems likely that the lack of strong horizontal edges meant that when an incorrect alignment was proposed and approved by the corner correspondence stage of VfA. Corners are deemed as inliers based on whether a given shift puts them on top of each other, not based on any kind of descriptor based on the visual appearance

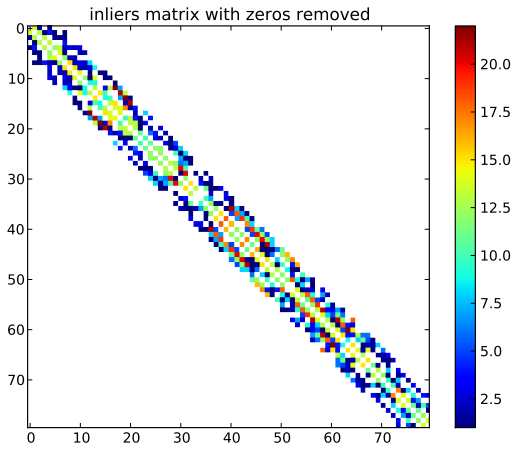


Fig. 13: number of inliers from grating sequence

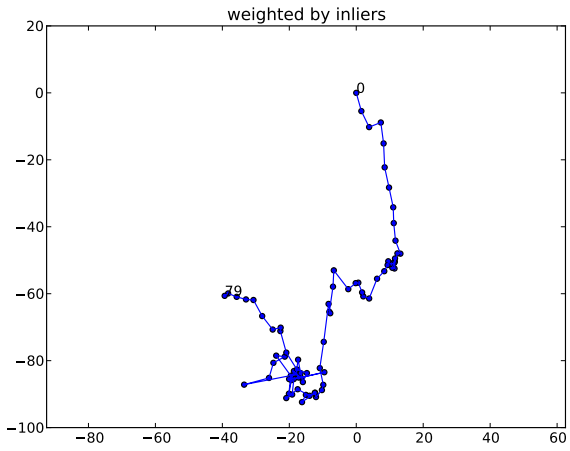


Fig. 15: Regularized camera locations for grating, weighted by inlier count. 0 is first frame, 79 is final frame.

of whatever was originally detected as a corner.

In this situation potentially even adding corner descriptors to the algorithm would not remedy the situation, because it seems like most corners are liable to be detected on either the drain or the two grooves next to it, and any hypothetical descriptor computed on these locations is likely to be visually similar to another descriptor computed somewhere else on the drain / groove. The best movement cues in this scene are the water stains on the floor, which are non-repeating, but these are not captured well by the VFA digest.

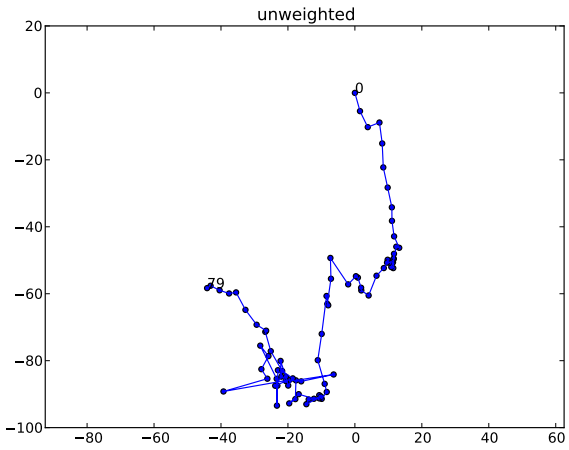


Fig. 14: Regularized camera locations for grating, unweighted. 0 is first frame, 79 is final frame.

2 Further Swipe Mosaic Results

2.1 Synthetic Satellite footage dataset

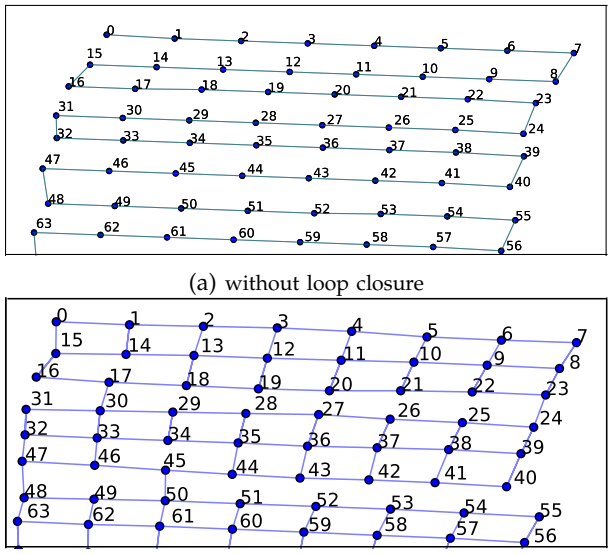
As well as the real video sequences, we converted a time-lapse video of the Earth recorded from the International Space Station into a Swipe Mosaic. We first constructed an intermediate video by cropping out a thin horizontal strip from the bottom of each image and splitting each strip into eight overlapping images. A virtual camera was moved back and forth along the strips, moving forward in time upon reaching the end, to give these cut up frames a nominal temporal ordering. We run pairwise prediction, regularization and translational loop closure on this sequence, and know that the images should ideally be estimated to form a regular rectangular grid.

It is hoped each image would know from the translational RRF that their neighbors on the same strip were at a purely horizontal offset, and neighbors on a different strip were at a purely vertical offset. The output of the first regularization step is shown in Fig. 17a. The arrangement is approximately what we would have hoped for, but the locations as a whole "lean" to one side. This can be explained by noting that for the images at either end of the strip, when the virtual camera moves "up" or "down", the overlapping pixel data between the image at either end of this link will actually move diagonally, because all the earth's surface appears to be moving away from the focus of expansion. Because the strips were not symmetrically cropped (the main goal of the cropping was to remove the visible parts of the ISS which appeared in the frame, of which there was more on one side) we see that the "up-down" links such as $(7, 15)$ push the entire system to the left to a greater extent than the links on the other side such as $40, 48$.

Ideally, our loop closure step should (with slightly modified thresholds to account for the shorter loops present in this artificial scene versus a real scene) detect loop points between every image and its corresponding vertical neighbors, *i.e.* link $(0, 15), (1, 14), (2, 13)$ *etc.* After incorporating a pairwise prediction from each and re-regularizing, we would hope to see the same overall grid structure, but with less of the horizontal skew visible in Fig. 17a. The result of automatic loop closure is shown in Fig. 17b. Loop points have been found in the majority

Fig. 16: Example images from ISS sequence, from the right hand edge of the image grid. Note how structures travel diagonally down to the right between frames - this is due to the curvature of the earth from the ISS' vantage point.

of places we hoped to see them. The output is not perfect but the transitions between, for example, images 3, 12 and 19 are closer to vertical than before the loop closure, and so are correspondingly improved in the visualization For this dataset, vertical would be the ideal answer. Even with the imperfect results, the ISS sequence is easy to navigate as a Swipe Mosaic.



(a) without loop closure

(b) with loop closure

Fig. 17: Loop closure on ISS. For this sequence only, the loop point detection parameter was set to 10 to allow for shorter loops. Only locations of the initial 64 frames are shown for clarity.

3 Implementation Details

3.1 Gabor Filter Bank

As mentioned in the the *Feature Computation* section of the main paper, a bank of Gabor filters are used as part of the feature computation process. Each filter is computed from the product of a Gaussian and a sinusoid, according to (1).

$$g(x, y; \lambda, \theta, \sigma, \gamma) = \exp\left(-\frac{\hat{x}^2}{2\sigma^2} - \frac{\hat{y}^2}{2\sigma_y^2}\right) \exp\left(\frac{2\pi\hat{x}}{\lambda}\right) \quad (1)$$

$$\hat{x} = x\cos\theta + y\sin\theta \quad (2)$$

$$\hat{y} = -x\sin\theta + y\cos\theta \quad (3)$$

$$\sigma_y = \frac{\sigma}{\gamma} \quad (4)$$

$$(5)$$

λ represents the wavelength of the sinusoid, θ is the orientation of the sinusoid (the orientation parameters allows the detection of multimodal ridges at different angles), σ represents the standard deviation of the Gaussian, and γ controls how this standard deviation varies in the x and y directions (ie creating an elliptical function). The ranges of values used for these parameters is specified in 1. For each configuraton of parameters, the filter is created as wide (in pixels) as necessary to encompass 3 standard deviations for the Gaussian.

λ	θ	σ	γ
100	0	4	1
10	$0, \frac{\pi}{4}, \ldots, \frac{7}{4}\pi$	2	1
10	$0, \frac{\pi}{4}, \ldots, \frac{7}{4}\pi$	2	0.5
10	$0, \frac{\pi}{4}, \ldots, \frac{7}{4}\pi$	3	1
10	$0, \frac{\pi}{4}, \ldots, \frac{7}{4}\pi$	3	0.5

TABLE 1: Parameters used to generate the Gabor filter bank.

References

[1] C. Wu, "SiftGPU: A GPU implementation of scale invariant feature transform (SIFT)," http://cs.unc.edu/ ccwu/siftgpu, 2007.

[2] C. Wu, S. Agarwal, B. Curless, and S. Seitz, "Multicore bundle adjustment," *CVPR*, 2011.

[3] A. Adams, N. Gelfand, and K. Pulli, "Viewfinder alignment," *Eurographics*, 2008.

[4] R. Szeliski, "Image alignment and stitching: a tutorial," *Found. Trends. Comput. Graph. Vis.*, 2006.

[5] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *IJCV*, 2004.