24 Dec 2016
E. Spalding

Finding a dewarp and orientation solution for LMIRCam/NOMIC

This procedure was followed to produce a dewarp solution and orientation determination for LMIRCam in Dec 2016. Feel free to adapt this procedure as you like. I recommend starting by making the following directory tree to include raw FITS files, and ones that are written out at intermediary steps:

`/asterism/` (data for finding the plate scale and orientation)
`/pinhole/` (data for finding the dewarp solution)

Further down, make

```
/asterism/rawData/
/asterism/processedData/
/pinhole/rawData/
/pinhole/darkSubtBadPixCorrect/
```

and, since the asterism data takes a few more intermediary steps,

```
/asterism/processedData/step01_darkSubtBadPixCorrect/
/asterism/processedData/step02_dewarped/
/asterism/processedData/step03_derotate/
/asterism/processedData/step04_ditherMedians/
```

Put the raw readouts into `/pinhole/rawData/` or `/asterism/rawData/`. Use your own custom dark-subtract and bad-pixel correction code and put those frames into the `*darkSubtBadPixCorrect/` directories. Naturally, as we walk through the code, be sure to check that all pathnames are right (n.b. the `__init__` file).

1 Dewarp solution: `find_dewarp_solution.py`

Requirements: You will need a detector image of a collection of well-sampled points, such as from a pinhole grid (Fig. 1).

1.1 The idea

We want to find the polynomial coefficients that map between empirical pinhole locations and an idealized grid. We use a direct transliteration of IDL's

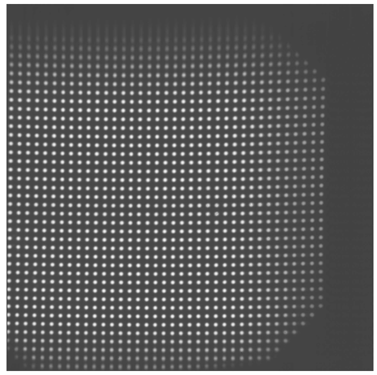


Figure 1: An example of what a pinhole-grid-illuminated LMIRCam readout should look like. Make sure not to saturate the pinholes, so as to facilitate centroiding.

`polywarp` procedure, which finds the coefficients $K_x^{(i,j)}$ and $K_y^{(i,j)}$ in the following polynomial mapping among (x, y) coordinates between the warped and ideal readouts:

$$x_i = \sum_{i=0}^{N} \sum_{j=0}^{N} K_x^{(i,j)} x_o^{(j)} y_o^{(i)} \tag{1}$$

$$\underbrace{y_i}_{\text{warped}} = \sum_{i=0}^{N} \sum_{j=0}^{N} K_y^{(i,j)} \underbrace{x_o^{(j)} y_o^{(i)}}_{\text{dewarped}} \tag{2}$$

Note which sides of the mapping represent the 'warped' and 'dewarped' coordinates in this application, which may be opposite to what one may expect intuitively, or from the IDL documentation on `polywarp`. Let's see why we do it this way by plunging into the functions called within the `find_dewarp_solution.py` script.

1.2 `make_dewarp_coords()`

Within the script `find_dewarp_solution.py`, you will see some functions and arrays appear in the first section of the code that you may have to run through a couple times so that you can tweak the function inputs to values that are optimal for your data. (See comments in the code for details.) It's also probably good to mask pinholes in the heavily vignetted region of the array (Fig. 2).

Once that's done, run the script again so that it runs past the function `make_dewarp_coords()`. This finds the aforementioned coefficients by solving a least-squares problem via

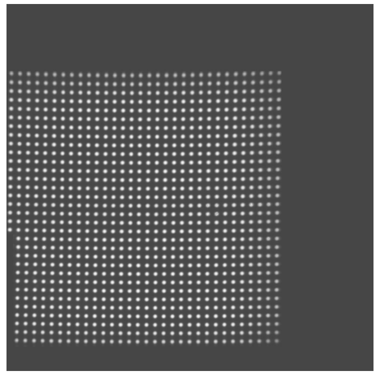


Figure 2: The actual pinhole image I used, with regions with vignetting and partially cut-off pinholes set to a constant.

Moore-Penrose pseudoinverse matrices. (I find J. Stone's condensed description of this to be helpful.) Schematically, what is being done is shown in cartoon form in Fig. 3.

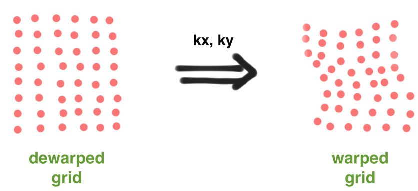


Figure 3: The idea behind `make_dewarp_coords()`.

1.3 `dewarp_with_precomputed_coords()`

The next function takes the raw image, pastes the warped coordinates onto it, and then smooths everything out by resampling the image point-by-point over the entire image space, interpolating as needed when the coordinates are not at integer values (Fig. 4).
As a check, closely compare the pinhole grid images before and after (Fig. 5).

1.4 Dewarp science images

This step in the code is self-explanatory. Just make sure the pathname-associated strings are correct.

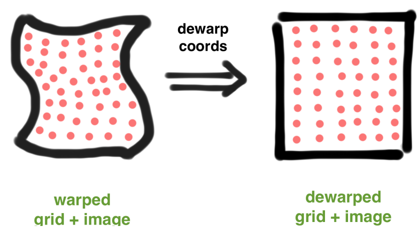


Figure 4: The idea behind `dewarp_with_precomputed_coords()`.

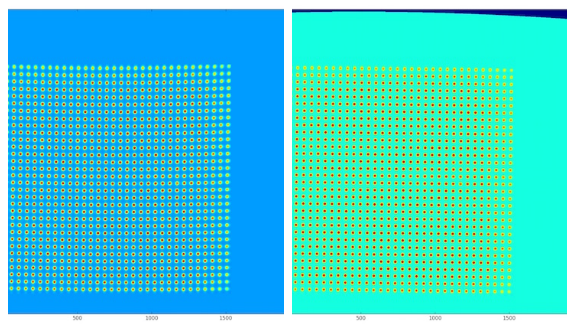


Figure 5: Pinholes before the dewarping (left) and afterwards (right). Color scale is arbitrary.

1.5 Barb (quiver) plot

The last part of the script makes a lovely barb plot, putting evenly-spaced vectors over the array to show the directions that points on the readouts have to be stretched in order to dewarp it (Fig. 6).

2 Orientation: `find_asterism_star_locations.py`, `find_asterism_star_locations.py`

Requirements: Images of a well-characterized stellar field, such as the Trapezium Cluster, at a number of dither positions. Images must already be dewarped and parallactic-angle-derotated.

2.1 The idea

We find star positions in pixel space, and use baselines between every possible pair of stars to find their separations and angles relative to north at PA=0. The separations are divided by their corresponding values in arcseconds from

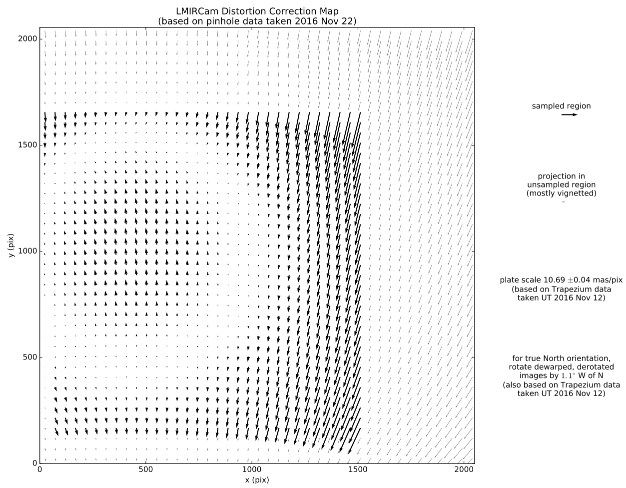


Figure 6: The barb plot. The doughnut shape is real, but is likely stamped into the optical beam by an optical element, not the array. Note also that this image includes rotation information off to the side, which I added in after determining the orientation (see below).

previously-constrained 'true' astrometric values to find the plate scale, and the angles are compared with their 'true' counterparts to find the residual angular offset of the detector. Note that in one frame of N_s stars, the total number of baselines among stellar pairs is "N_s choose 2":

$$N_b = \binom{N_s}{2} \equiv \frac{N_s!}{2!(N_s - 2)!} = \frac{N_s(N_s - 1)}{2} \tag{3}$$

2.2 Find stars in pixel space: `find_asterism_star_locations.py`

The pre-requisite dewarping of images is performed in `find_dewarp_solution.py`. But you also need to derotate them based on their parallactic angle. For this I used a quick IDL script that read the headers and used the IDL function `rot`. (I included a FYI copy of my script `derotate_trapezium_data_ut_2016_11_12.pro`.) However you do it, take the dewarped FITS files (which are now residing in `/step02_dewarped/`, derotate them, and write out the results to the directory `/step03_derotate/`.

Now take the median of each dither position. Dump these medians into `/step04_ditherMedians/`. (Since calibration-related images are often taken during mediocre conditions, it is important to take many images at a given dither position.) Overlay the resultant dither medians (Adobe Photoshop is very useful for this), mark any visible stars, and then cross-check them with a known astrometric source. In the case of the Trapezium Cluster, one can use the images and Table 1 in Close+ 2012 *ApJ* 749:180. In Fig. 7, I labeled stars using the conventions in Close+ 2012, and used my own Greek lettering if they were without label.

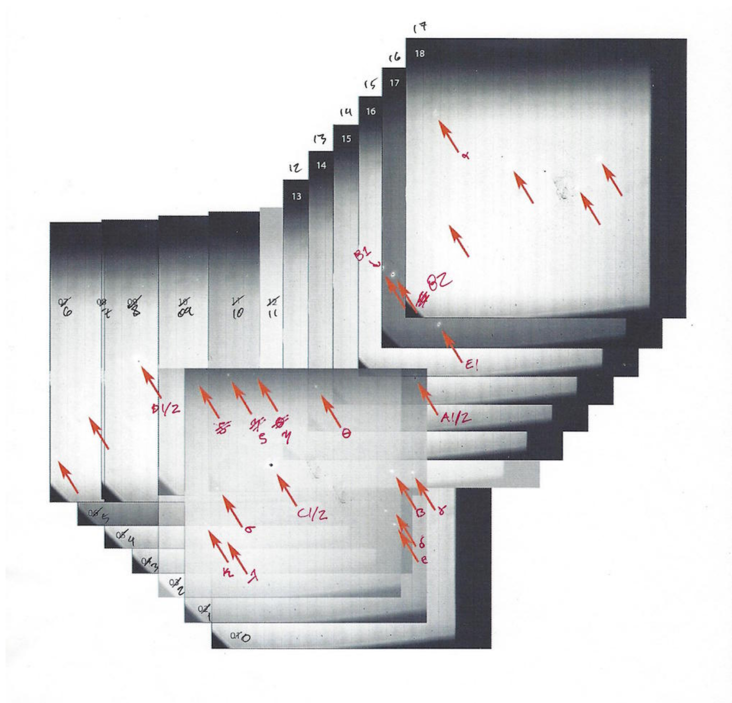


Figure 7: A mosaic of images, each representing a different dither position, using different transparencies so as to overlay the stars as closely as possible. The images here have not been derotated, but they were all taken near transit within about 10 degrees of each other. The point here is just to have a handy visual for cross-checking the stars. ('D1/2' should actually just be 'D1').

The script `find_asterism_star_locations.py` takes the intermediary step of determining star locations in pixel space, and printing locations in pixel space to the screen. Check each centroid manually in the plot, to see if it's a real star or not (Fig. 8). Copy the true positive locations in pixel space that are returned in the Terminal, and populate the dictionaries in the script `find_plate_scale_and_orientation.py`.

2.3 Find plate scale, angle offset: `find_plate_scale_and_orientation.py`

This standalone script contains a long, unwieldy series of dictionaries, which you need to painstakingly populate as mentioned above. Once this is done, it will print the angular differences and plate scales corresponding to every possible

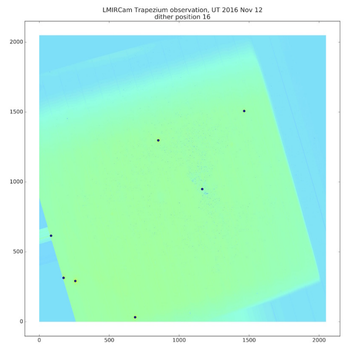


Figure 8: An example image returned by `find_asterism_star_locations.py` to allow the user to check found centroids, and pick the true positives printed in the Terminal.

baseline between a pair of stars, and finish off with the net detector plate scale and angular offset, which will be printed to the screen with 1-sigma boundaries, and will appear in a plot (Fig. 12).

2.4 Barb plot with arbitrary K_x, K_y inputs: `make_barb_plot_kxky_coeffs.py`

This script is available for making a separate barb plot from the mapping coefficients only, such as those available in Maire+ 2015 *A&A* 576 A133 (Fig. 13).

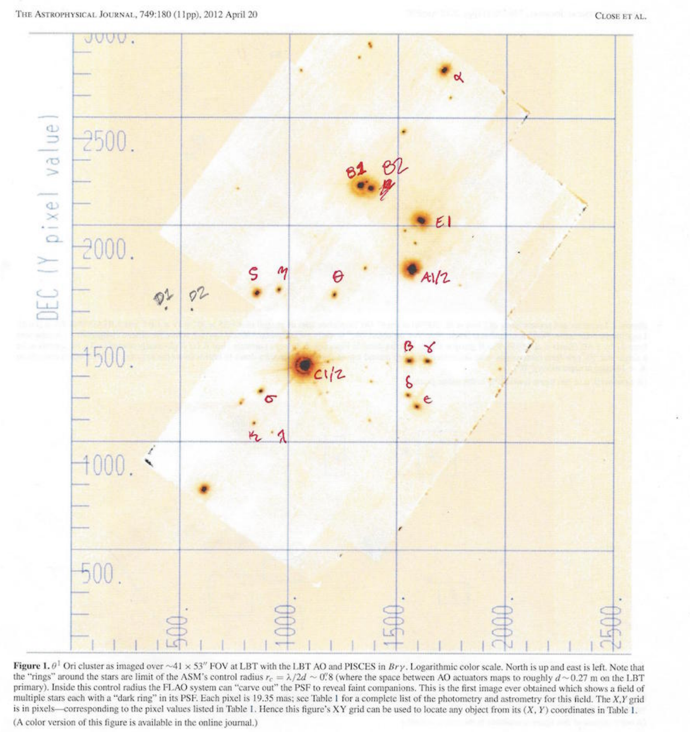


Figure 9: An image from Close+ 2012, used for cross-checking the stars in our mosaic. (At least one of the labels is wrong. Find it!)

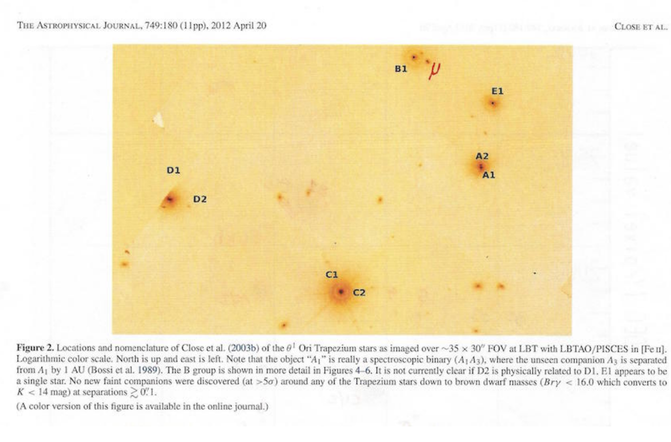
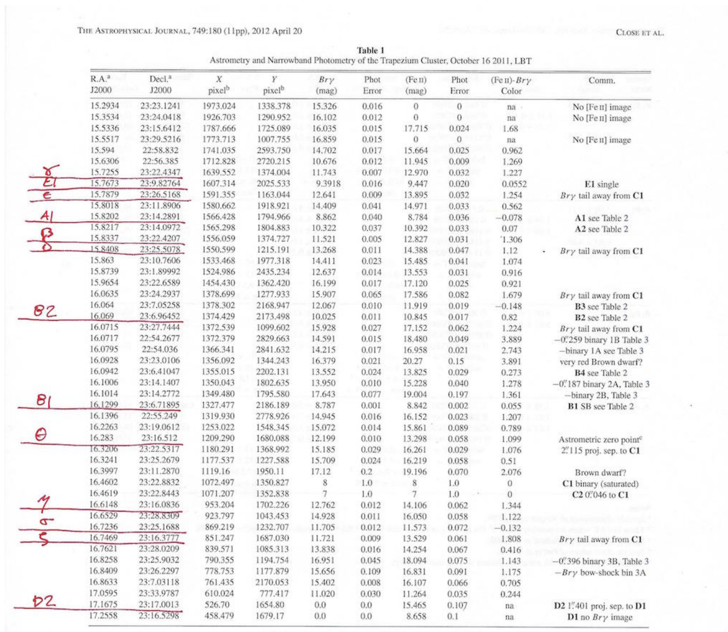


Figure 10: Another relevant image from Close+ 2012.



THE ASTROPHYSICAL JOURNAL, 749:180 (11pp), 2012 April 20 CLOSE ET AL.

Table 1
Astrometry and Narrowband Photometry of the Trapezium Cluster, October 16 2011, LBT

	R.A.[a] J2000	Decl.[a] J2000	X pixel[b]	Y pixel[b]	$Br\gamma$ (mag)	Phot Error	(Fe II) (mag)	Phot Error	(Fe II)-$Br\gamma$ Color	Comm.
	15.2934	23:23.1241	1973.024	1338.378	15.326	0.016	0	0	na	No [Fe II] image
	15.3534	23:24.0418	1926.703	1290.952	16.102	0.012	0	0	na	No [Fe II] image
	15.5336	23:15.6412	1787.666	1725.089	16.035	0.015	17.715	0.024	1.68	
	15.5517	23:29.5216	1773.713	1007.755	16.859	0.015	0	0	na	No [Fe II] image
	15.594	22:58.832	1741.035	2593.750	14.702	0.017	15.664	0.025	0.962	
	15.6306	22:56.385	1712.828	2720.215	10.676	0.012	11.945	0.009	1.269	
γ	15.7255	23:22.4347	1639.552	1374.004	11.743	0.007	12.970	0.032	1.227	
E1	15.7673	23:9.82764	1607.314	2025.533	9.3918	0.016	9.447	0.020	0.0552	**E1** single
ε	15.7879	23:26.5168	1591.355	1163.044	12.641	0.009	13.895	0.032	1.254	$Br\gamma$ tail away from **C1**
	15.8018	23:11.8906	1580.662	1918.921	14.409	0.041	14.971	0.033	0.562	
A1	15.8202	23:14.2891	1566.428	1794.966	8.862	0.040	8.784	0.036	−0.078	**A1** see Table 2
	15.8217	23:14.0972	1565.298	1804.883	10.322	0.037	10.392	0.033	0.07	**A2** see Table 2
β	15.8337	23:22.4207	1556.059	1374.727	11.521	0.005	12.827	0.031	1.306	
δ	15.8408	23:25.5078	1550.599	1215.191	13.268	0.011	14.388	0.047	1.12	$Br\gamma$ tail away from **C1**
	15.863	23:10.7606	1533.468	1977.318	14.411	0.023	15.485	0.041	1.074	
	15.8739	23:1.89992	1524.986	2435.234	12.637	0.014	13.553	0.031	0.916	
	15.9654	23:22.6589	1454.430	1362.420	16.199	0.017	17.120	0.025	0.921	
	16.0635	23:24.2937	1378.699	1277.933	15.907	0.065	17.586	0.082	1.679	$Br\gamma$ tail away from **C1**
	16.064	23:7.05258	1378.302	2168.947	12.067	0.010	11.919	0.019	−0.148	**B3** see Table 2
B2	16.069	23:6.96452	1374.429	2173.498	10.025	0.011	10.845	0.017	0.82	**B2** see Table 2
	16.0715	23:27.7444	1372.539	1099.602	15.928	0.027	17.152	0.062	1.224	$Br\gamma$ tail away from **C1**
	16.0717	22:54.2677	1372.379	2829.663	14.591	0.015	18.480	0.049	3.889	−0.″259 binary 1B Table 3
	16.0795	22:54.036	1366.341	2841.632	14.215	0.017	16.958	0.021	2.743	−binary 1A see Table 3
	16.0928	23:23.0106	1356.092	1344.243	16.379	0.021	20.27	0.15	3.891	very red Brown dwarf?
	16.0942	23:6.41047	1355.015	2202.131	13.552	0.024	13.825	0.029	0.273	**B4** see Table 2
	16.1006	23:14.1407	1350.043	1802.635	13.950	0.010	15.228	0.040	1.278	−0.″187 binary 2A, Table 3
	16.1014	23:14.2772	1349.480	1795.580	17.643	0.077	19.004	0.197	1.361	−binary 2B, Table 3
B1	16.1299	23:6.71895	1327.477	2186.189	8.787	0.001	8.842	0.002	0.055	**B1** SB see Table 2
	16.1396	22:55.249	1319.930	2778.926	14.945	0.016	16.152	0.023	1.207	
	16.2263	23:19.0612	1253.022	1548.345	15.072	0.014	15.861	0.089	0.789	
θ	16.283	23:16.512	1209.290	1680.088	12.199	0.010	13.298	0.058	1.099	Astrometric zero point[c]
	16.3206	23:22.5317	1180.291	1368.992	15.185	0.029	16.261	0.029	1.076	2.″115 proj. sep. to **C1**
	16.3241	23:25.2679	1177.537	1227.588	15.709	0.024	16.219	0.058	0.51	
	16.3997	23:11.2870	1119.16	1950.11	17.12	0.2	19.196	0.070	2.076	Brown dwarf?
	16.4602	23:22.8832	1072.497	1350.827	8	1.0	8	1.0	0	**C1** binary (saturated)
	16.4619	23:22.8443	1071.207	1352.838	7	1.0	7	1.0	0	**C2** 0.″046 to **C1**
μ	16.6148	23:16.0836	953.204	1702.226	12.762	0.012	14.106	0.062	1.344	
	16.6529	23:28.8309	923.797	1043.453	14.928	0.011	16.050	0.058	1.122	
σ	16.7236	23:25.1688	869.219	1232.707	11.705	0.012	11.573	0.072	−0.132	
ζ	16.7469	23:16.3777	851.247	1687.030	11.721	0.009	13.529	0.061	1.808	$Br\gamma$ tail away from **C1**
	16.7621	23:28.0209	839.571	1085.313	13.838	0.016	14.254	0.067	0.416	
	16.8258	23:25.9032	790.355	1194.754	16.951	0.045	18.094	0.075	1.143	−0.″396 binary 3B, Table 3
	16.8409	23:26.2297	778.753	1177.879	15.656	0.109	16.831	0.091	1.175	−$Br\gamma$ bow-shock bin 3A
	16.8633	23:7.03118	761.435	2170.053	15.402	0.008	16.107	0.066	0.705	
	17.0595	23:33.9787	610.024	777.417	11.020	0.030	11.264	0.035	0.244	
D2	17.1675	23:17.0013	526.70	1654.80	0.0	0.0	15.465	0.107	na	**D2** 1.″401 proj. sep. to **D1**
	17.2558	23:16.5298	458.479	1679.17	0.0	0.0	8.658	0.1	na	**D1** no $Br\gamma$ image

Figure 11: Origin of the 'true' coordinates: Table 1 from Close+ 2012.

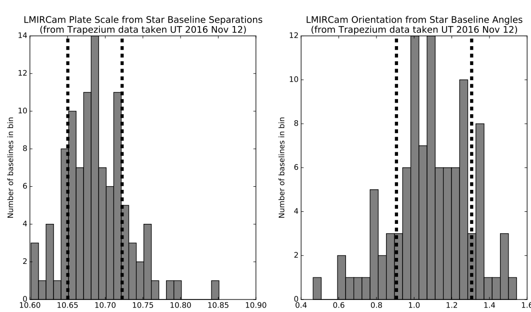


Figure 12: Histograms returned by `find_plate_scale_and_orientation.py`. Dashed vertical lines indicate 1-sigma bounds.

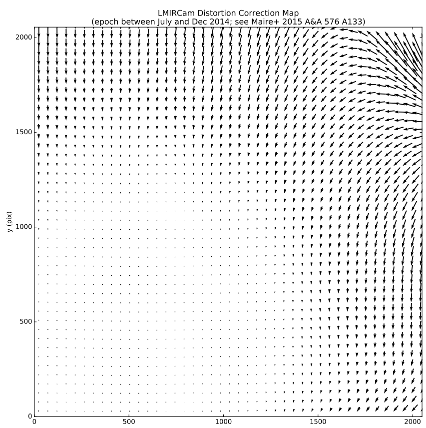


Figure 13: A distortion map made from 2014-vintage mapping coefficients.