LARGE SYNOPTIC SURVEY TELESCOPE

Large Synoptic Survey Telescope (LSST)

# jointcal: Simultaneous Astrometry & Photometry for thousands of Exposures with Large CCD Mosaics

**John Parejko (University of Washington), Pierre Astier (LPNHE/IN2P3/CNRS Paris)**

**DMTN-036**

**Latest Revision: 2017-09-18**

**DRAFT**

## Abstract

The jointcal package simultaneously optimizes the astrometric and photometric calibrations of a set of astronomical images. In principle and often in practice, this approach produces distortion and thoroughput models which are more precise than when fitted independently. This is especially true when the images are deeper than the astrometric reference catalogs. In the "Astromatic" software suite, this simultaneous astrometry functionality is fulfilled by "SCAMP". The code we describe here has similar aims, but follows a slightly different route. Jointcal is built on top of the the LSST Data Management software stack.

# Change Record

| Version | Date | Description | Owner name |
|---|---|---|---|
| 1 | 2017-09-18 | Initial release. Moved from jointcal repo. | John Parejko |

# Contents

# jointcal: Simultaneous Astrometry & Photometry for thousands of Exposures with Large CCD Mosaics

## 1 Introduction

With deep astronomical images, it is extremely common that the relative astrometry and photometry between images is considerably more precise than the accuracy of external catalogs, where "more precise" can be as large as two orders of magnitude. For applications where the quality of relative astrometry is important or vital, it is important to rely on some sort of simultaneous astrometry solution, if possible optimal in a statistical sense.

This package performs a least-squares fit to a set of images. Since it aims at statistical optimality, we maximize the likelihood of the measurements with respect to all unknown parameters required to describe the data. These parameters are mostly in two sets: the position on the sky of the objects in common between the images, and the mapping of each image to the sky. To these obvious parameters, one can add proper motions (where applicable), and parameters describing the differential effect of atmospheric refraction on the position of objects. It is clear that one cannot fit simultaneously the position on the sky and the mappings from CCD coordinates to the sky, without extra constraints: the "sky" coordinate system is then undefined, and one needs reference positions in order to fully define this frame. We use the GAIA catalog (Gaia Collaboration et al., 2016) as our reference catalog, and may supplement it with deeper data (e.g. PS1) where available.

SCAMP (Bertin, 2006) is the reference package for simultaneous astrometry in astronomy, at least for relative alignment of wide-field images prior to stacking. Regarding optimization, SCAMP follows a somewhat different route from ours: it does not optimize over the position of common objects but rather minimizes the distance between pairs of transformed measurements of the same object. This approach is not a maximum likelihood optimization, and is likely statistically sub-optimal. The main drawback of SCAMP in the context of LSST is the fact that it is a program and not a library, and hence not flexible regarding formats of images and catalogs. But since SCAMP has been used for almost a decade in production by various teams, the quality checking tools it provides should likely be reproduced in the context of our package. We provide residual ntuples and hope that the first serious users will contribute plotting tools.

Loading input catalogs and their metadata from disk is a large fraction of the total time. We

can save time by re-using the input catalog to fit a similar style of multi-component model for the relative and absolute photometry.

The plan of this note is as follows: we first sketch the algorithm (Section 3). We provide our least-squares formulation in Section 4, and describe the how we evaluate the derivatives with respect to the parameters. We then describe how we associate the measurements of each object in different exposures in Section 5.

## 2 Past work

A summary of past work on this topic will go here eventually...

- Photographic Plates (Eichhorn, 1960)
- SDSS übercal (Padmanabhan et al., 2008)
- Pan-Starrs übercal (Magnier et al., 2013)
- DECam WcsFit (Bernstein et al., 2017)

## 3 Algorithm flow

The algorithm assumes that the initial single-frame WCS fits of the input images are accurate to $\sim 1''$. Currently, the code properly interprets the SIP WCS's (relying on `lsst::afw::wcs`), with or without distortions. The code might handle transparently the "PV" encoding of distortions (used in SCAMP and Swarp), but lacks the IO's required to use this format. Note that in both instances, the WCS boils down to a polynomial 2D transform from CCD space to a tangent plane, followed by a gnomonic de-projection to the celestial sphere. The difference between formats lies into the encoding of the polynomial, but they map exactly the same space of distortion functions.

The algorithm can be roughly split into these successive steps:

1. load the input catalogs and 'rough' WCS's and catalogs, and select the sources to use in the fit (e.g. good centroids, unblended, high enough signal-to-noise) via the configured SourceSelectorTask.

2. Associate these catalogs, i.e. associate each detection of the same on-sky source, and associate those on-sky sources with a reference catalog (if provided).
3. Iteratively fit the model parameters and "true" on-sky values, clipping outliers at each iteration.
4. Output results.

# 4 Mathematical formalism

## 4.1 Definitions

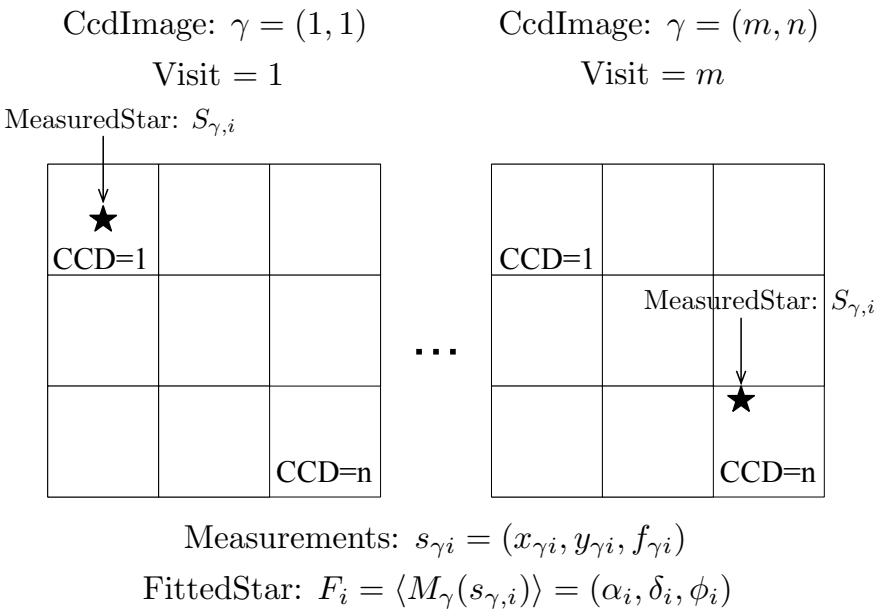


FIGURE 1: The relationship between MeasuredStars, CcdImages, and FittedStars.

We use the following notation in the mathematics that follows, with *CamelCased* words referring to the objects in the code. These terms are diagramed in Figure 1 on page 3.

- $\gamma$ is the *CcdImage* representing one exposure (*visit* in LSST terminology) on one CCD. The *CcdImage* contains metadata about that visit and CCD detector, as well as the catalog of sources that were selected for use in the fit;

- $S_{\gamma,i}$ is the position (pixels) and flux (instrument-flux counts) $(x_{\gamma i}, y_{\gamma i}, f_{\gamma i})$ of the *Measured-Star* on *CcdImage* $\gamma$ corresponding to the on-sky *FittedStar* $i$;
- $F_i$ is the (sky) position and calibrated-flux $(\alpha_i, \beta_i, \phi_i)$ of the star $i$, with some corresponding number of measurements represented by *MeasuredStar*s;
- $M_\gamma$ is the mapping for *CcdImage* $\gamma$ from pixels/instrument-flux $(x, y, f)$ to the tangent plane on the sky/calibrated-flux $(\alpha, \beta, \phi)$. This mapping may consist of models constrained across visits (different between CCDs; e.g. an affine model for each CCD position, assuming CCDs do not move), constrained across CCDs and visits (e.g. a 2d radial polynomial of the optics) or constrained across CCDs (different between visits; e.g. a 2d polynomial of the sky);
- $R_j$ refers to the (sky) position and (reference) calibrated-flux $(\alpha_j, \beta_j, \phi_j)$ of *RefStar* $j$.

In addition to the terms defined in the diagram above, we define the following terms for the measurement component,

- $P_\gamma$ is a projector from sidereal coordinate to some tangent plane; $P_\gamma$ is user-defined;
- $W_{\gamma,i}$ is the measurement weight of $M_\gamma(S_{\gamma,i})$, i.e. the inverse of the 2×2 covariance matrix (for astrometry), or the inverse of the transformed instrument-flux error;

and reference component,

- $P$ is some (user-provided) sky to tangent plane projector;
- $W_j$ is the weight matrix of the reference star, i.e. the inverse of the projected position error $P(R_j)$, or the inverse of the reference flux error.

## 4.2 Least-squares expression

The fit consists of minimizing, for photometry,

$$\chi^2 = \sum_{\gamma,i} [M_\gamma(S_{\gamma,i}) - F_i]^T W_{\gamma,i} [M_\gamma(S_{\gamma,i}) - F_i] \quad \text{(meas. terms)}$$
$$+ \sum_j [F_j - R_j]^T W_j [F_j - R_j] \quad \text{(ref. terms)} \quad (1)$$

and for astrometry, taking into account the projection from the sky to the tangent plane $P$:

$$\chi^2 = \sum_{\gamma,i} [M_\gamma(S_{\gamma,i}) - P_\gamma(F_i)]^T W_{\gamma,i} [M_\gamma(S_{\gamma,i}) - P_\gamma(F_i)] \quad \text{(meas. terms)}$$
$$+ \sum_j [P(F_j) - P(R_j)]^T W_j [P(F_j) - P(R_j)] \quad \text{(ref. terms)} \quad (2)$$

where the first line iterates on all *MeasuredStar* $\gamma, i$, from all *CcdImage* $\gamma$, and the second iterates on all *RefStar* $j$. In the first terms, the object at position $F_i$ is the one that was measured at position $S_{\gamma,i}$ in image $\gamma$. The association between *MeasuredStar*s, *FittedStar*s, and *RefStar*s is described in Section 5.

The measurement terms compare the measurement positions/fluxes to objects positions/fluxes (the relative astrometry/photometry), the reference terms compare object positions/fluxes to reference positions/fluxes (the absolute astrometry/photometry). We need these two sets of terms because not all objects $F_i$ in the first terms appear in the second terms: many objects in the images will not be in the reference catalogs, but those objects do help to constrain the mappings $M_\gamma$. In addition, with so many measurements of our sources, we may have better overall errors on the positions and fluxes than the reference catalog can provide.

The expressions above depend on two sets of parameters: the parameters defining the mappings $M$ and the on-sky positions/calibrated fluxes $F_i$. For a practical problem, this amounts to a very large number of parameters, which becomes tractable when one notes that every term in the $\chi 2$ is only linked to a small number of parameters. We exploit this feature to rapidly compute the gradient and the Hessian of the $\chi^2$ in order to find the minimum.

So far, we have not specified how we model the mappings $M$ nor how we choose the various projectors that appear in expression (2). The code has been written to allow the user to provide their own versions of both the model for mappings and the projection scheme. We however provide some implementations for both aspects that we discuss in the next two sections.

### 4.3 Minimization approach

The expressions for astrometry (eq. (2)) and photometry (eq. (1)) depend on two sets of parameters: the parameters $\eta$ defining the mappings $M_\gamma(S_{\gamma,i}) \equiv M_\gamma(\eta_\gamma, S_{\gamma,i})$, and the "true" calibrated fluxes $F_i$. We write the measurement and reference residuals, with their respective weights $W_{\gamma i}$ and $W_j$, as separate functions of their parameters, for photometry,

$$D_{\gamma i} = M_\gamma(S_{\gamma,i}) - F_i \tag{3}$$
$$D_j = F_j - R_j \tag{4}$$

and for astrometry,

$$D_{\gamma i} = M_\gamma(S_{\gamma,i}) - P_\gamma(F_i) \tag{5}$$
$$D_j = [P(F_j) - P(R_j)] \tag{6}$$

This results in the generalized $\chi^2$ expression (compare to eq. (2) and (1)),

$$\chi^2 = \sum_{\gamma,i} {D_{\gamma i}}^T W_{\gamma,i} D_{\gamma i} + \sum_j {D_j}^T W_j D_j \tag{7}$$

To minimize this $\chi^2$, we want to find the point in parameter space where the gradient $\nabla\chi^2 = d\chi^2/d\theta = 0$, where $\theta$ denotes the vector of parameters (of size $N_p$– see below). Applying the product rule and noting the symmetry of $D$ and $D^T$, we have

$$\frac{1}{2}\frac{d\chi^2}{d\theta} = \sum_{\gamma,i} {D_{\gamma i}}^T W_{\gamma,i} \nabla D_{\gamma i} + \sum_j {D_j}^T W_j \nabla D_j \tag{8}$$

where the $\nabla D$ matrices have size $2 \times N_p$ (for astrometry) and $1 \times N_p$ (for photometry):

$$\nabla D_{\gamma i} = \frac{d D_{\gamma i}}{d\theta} \tag{9}$$
$$\nabla D_j = \frac{d D_j}{d\theta} \tag{10}$$

We call the vector that gathers all the parameters to be fit during the minimization $\theta = (\eta_\gamma, F_i)$. This vector can easily exceed $N_p > 10^5$ entries. As an example with LSST (having a 189-CCD

camera), a mapping consisting of a 3rd order 2-D polynomial per visit (16 $\eta_k$ parameters per visit) and 100 viable sources per CCD with 15 visits (roughly a year of LSST observations of one sky location in one filter), $\theta$ will have $N_p = 283,740$ entries ($283,500$ $F_i$ parameters + 240 model parameters). However, the second derivative matrix, $d^2\chi^2/d\theta^2$, is very sparse, because there are no terms connecting $F_i$ and $F_j$ if $i \neq j$, and depending on how the mappings are parametrized, a set of $\eta_\gamma$ parameters could be connected (in the second derivative matrix) to only a small set of $F_j$'s. So, we can search for the minimum $\chi^2$ using methods involving the second derivative matrix, taking advantage of its sparseness.

We are looking for the offset to the starting parameters that zeros the gradient, so we can Taylor expand about that starting parameter vector $\theta_0$,

$$0 = \frac{d\chi^2}{d\theta}(\theta_0 + \delta\theta) = \frac{d\chi^2}{d\theta}(\theta_0) + \frac{d^2\chi^2}{d\theta^2}(\theta_0)\delta\theta + O(\delta\theta^2)$$

and solve for the offset $\delta\theta$ that zeroes it to first order,

$$\left[\frac{d^2\chi^2}{d\theta^2}(\theta_0)\right]\delta\theta = -\frac{d\chi^2}{d\theta}(\theta_0) \quad (11)$$

$$H\delta\theta = -\nabla\chi^2 \quad (12)$$

where we have identified the second derivative matrix with the Hessian matrix $H$. Working from eq. (8), we can write the second derivative matrix (the Hessian) as:

$$\frac{1}{2}\frac{d^2\chi^2}{d\theta^2} = \sum_{\gamma,i} \nabla {D_{\gamma i}}^T W_{\gamma,i} \nabla D_{\gamma i} + \sum_j \nabla {D_j}^T W_j \nabla D_j \quad (13)$$

where we have neglected the second derivatives of the residual vectors $D$. This second derivative matrix is by construction symmetric, and hence the parameter offsets (defined in eq. (11)) can be evaluated using the (fast) Cholesky $LDL^T$ factorization (see 4.9). If possible, mappings $M_\gamma(\eta_\gamma, S)$ linear with respect to their parameters $\eta_\gamma$, for example polynomials, are to be favored because the second derivatives will no longer depend on that parameter. If the problem were non-linear (more precisely, if the second derivative varies rapidly) we would have to implement a line search to minimize $\chi^2(\theta_0 + \lambda \times \delta\theta)$ over $\lambda$.

Returning to the weights, we note that the matrices $W_{\gamma,i}$ are positive-definite and thus they have square roots (e.g. the Cholesky square root) and can be written as: $W_{\gamma,i} = \alpha_{\gamma i}^T \alpha_{\gamma i}$. Defining

$K_{\gamma i} = \alpha_{\gamma i} D_{\gamma i}$, the Hessian expression becomes:

$$\frac{1}{2}\frac{d^2\chi^2}{d\theta^2} = \sum_{\gamma,i} {K_{\gamma i}}^T K_{\gamma i} + \sum_j {K_j}^T K_j \tag{14}$$

The sums present in this expression can be performed using matrix algebra; we concatenate all the $K$ matrices into a single large, sparse matrix (the Jacobian),

$$J \equiv \left[\{K_{\gamma i}, \forall \gamma, i\}, \{K_j, \forall j\}\right]$$

and thus we simply have

$$\frac{1}{2}\frac{d^2\chi^2}{d\theta^2} = J^T J$$

In the code, we take advantage of the fact that each term of the $\chi^2$ only depends on a small number of parameters. The *Model::getMappingIndices* method allows us to rapidly collect the indices of these parameters, and we evaluate the $D$ matrices at these indices only, as all other indices are zero.

The computation of the Jacobian and the gradient is performed in the respective *PhotometryFit* and *AstrometryFit* classes. The methods *leastSquareDerivativesMeasurement* and *leastSquareDerivativesReference* compute the contributions to the Jacobian and gradient of the $\chi^2$ from the measurement terms and the references terms respectively. In these routines, the Jacobian is represented as a list of *Eigen::Triplets* $(i, j, J_{ij})$ describing its elements. This list is then transformed into a representation of sparse matrices suitable for algebra, and in particular suitable to evaluate the product $J^T J$. Once we have evaluated $H \equiv J^T J$, we can solve eq. (12) using a Cholesky factorization. For sparse linear algebra, the Cholmod and Eigen packages provide the required functionality. It turns out that for the mappings we have currently employed, the calculation of $J^T J$ and the factorization are the most CPU intensive parts of the calculations, and there is hence not much to be gained in speeding up the calculation of derivatives. For the factorization, we have tried both Eigen and Cholmod (via the Eigen interface) and their speeds differ by less than 10%.

### 4.4 Photometry example

As an illustrative example, we will work through a particular photometry mapping, consisting of a constant zero-point per CCD ($f_0$: the CCD's filter response) and an $(n+m)$th order 2-D Chebyshev polynomial ($\sum a_{j,k} T_j(u) T_k(v)$: the optics+sky response, where $(u(x,y), v(x,y))$ are the

focal plane coordinates of pixel $(x, y)$ on a given CCD) per visit. Thus, the mapping will be

$$M_{\gamma i}(\eta, S_{\gamma i}) = M_{CCD}(f_0^{-1}, f_{\gamma i}) M_{visit}(a_{j,k}, x_{\gamma i}, y_{\gamma i}) = f_{\gamma i}[f_0]^{-1} \sum_{j=0}^{j=n} \sum_{k=0}^{k=m} a_{j,k} T_j(u_{\gamma i}) T_k(v_{\gamma i}) = \phi_{\gamma i}$$

where we will fit $f_0^{-1}$ instead of $f_0$ in order to simplify the derivatives (with respect to $\eta = (f_0^{-1}, a_{j,k} \forall j, k)$). Computing those derivatives gives us:

$$\nabla D_{\gamma i} = (\frac{\partial D_{\gamma i}}{\partial f_0^{-1}}, \frac{\partial D_{\gamma i}}{\partial a_{0,0}}, \ldots, \frac{\partial D_{\gamma i}}{\partial a_{n,m}}, \frac{\partial D_{\gamma i}}{\partial F_i})$$

where, for the measurement terms we have (recall eq. (3)),

$$\begin{aligned} \frac{\partial D_{\gamma i}}{\partial f_0^{-1}} &= f_{\gamma i} M_{visit}(a_{j,k} x_{\gamma i}, y_{\gamma i}) \\ \frac{\partial D_{\gamma i}}{\partial a_{j,k}} &= f_{\gamma i} f_0^{-1} T_{jk}(u_{\gamma i}, v_{\gamma i}) \\ \frac{\partial D_{\gamma i}}{\partial F_i} &= -1 \end{aligned}$$

and for the reference terms (recall eq. (4)),

$$\nabla D_j = \frac{\partial D_j}{\partial F_j} = 1$$

This model is degenerate to multiplying by a scale factor: $M_{CCD} \to a M_{CCD}, M_{visit} \to a^{-1} M_{visit}$. This degeneracy is not removed by the reference catalog. To break this degeneracy, we hold fixed one CCD's $f_0^{-1}$ (chosen to be the CCD closest to the center of the focal plane), and fit all other CCD's relative to that.

## 4.5 The astrometric distortion model

The routines in the *AstrometryFit* class do not really evaluate the derivatives of the mappings, but rather defer those to other classes. The main reason for this separation is that one could conceive different ways to model the mappings from pixel coordinates to the tangent plane, and the actual model should be abstract in the routines accumulating gradient and Jacobian. The class *AstrometryModel* is an abstract class aiming at connecting generically the fitting routines to actual models. We have so far coded two of these models:

- *SimplePolyModel* implements one polynomial mapping per input *CcdImage* (i.e. Calexp).
- *ConstrainedPolyModel* implements a model where the mapping for each CcdImage is a composition of a polynomial for each CCD and a polynomial for each exposure. For one of the exposures, the mapping should be fixed or the model is degenerate.

For example, if one fits 10 exposures from a 36-CCD camera, there will be $10 \times 36$ polynomials to fit with the first model, and $10 + 36$ with the second model. The *ConstrainedPolyModel* assumes that the focal plane of the instrument does not change across the data set. We could consider coding a model made from one *ConstrainedPolyModel* per set of images for which the instrument can be considered as geometrically stable. This is similar to how Scamp models the distortions.

In both of these models, we have used standard polynomials in 2 dimensions rather than an orthogonal set (e.g. Legendre, Laguerre, ...) because regular polynomials are easy to compose (i.e. one can easily compute the coefficients of P(Q(X)) ), and they map exactly the same space as the common orthogonal sets. We have taken care to normalize the input coordinates (mapping the range of fitted data over the $[-1, 1]$ interval), in order to alleviate the well-know numerical issues associated to fitting of polynomials.

## 4.6 Choice of projectors

In the least squares expression (2), the residuals of the measurement terms read:

$$D_{\gamma i} = M_\gamma(S_{\gamma i}) - P_\gamma(F_i)$$

If the coordinates $F_i$ are sidereal coordinates, the projector $P_\gamma$ determine the meaning of the mapping $M_\gamma$. If one is aiming at producing WCS's for the image, it seems wise to choose for $P_\gamma$ the projection used for the envisioned WCS, so that the mapping $M_\gamma$ just describes the transformation from pixel space to the projection plane. For a SIP WCS, one will then naturally choose a gnomonic projector, so that $M_\gamma$ can eventually be split into the "CD" matrix and the SIP-specific higher order distortion terms (see Section A for a brief introduction to WCS concepts).

So, the choice of the projectors involved in the fit are naturally left to the user. This is done via a virtual class *ProjectionHandler*, an instance of which has to be provided to the *AstrometryFit* constructor. There are obviously ready-to-use *ProjectionHandler* implementations which should suit essentially any need. For the standard astrometric fit aiming at setting WCS's, we

provide the *OneTPPerShoot* derived class, which implements a common projection point for all chips of the same exposure. It is fairly easy to implement derived classes with other policies.

The choice of the projector appearing in the reference terms of eq. (2) is not left to the user because we could not find a good reason to provide this flexibility, and we have implemented a gnomonic projection. We use a projector there so that the comparison of positions is done using an Euclidean metric.

## 4.7 Proper motions and atmospheric refraction

The expression (2) above depends on two sets of parameters: the parameters defining the mappings and the positions $F_k$. This expression hides two details implemented in the code: accounting for proper motions and differential effects of atmospheric refraction.

Proper motions can be accounted for to predict the expected positions of objects and even be considered as fit parameters. At the moment we neither have code to detect that some (presumably stellar) object is moving, nor code to ingest proper motions from some external catalog. Each *FittedStar* has a flag that says whether it is affected by a proper motion and the proper motion parameters can all be fitted or not (see Section 4.10).

The code allows to account for differential chromatic effects of atmospheric refraction, i.e. the fact that objects positions in the image plane are shifted by atmospheric refraction in a way that depends on their color. The shift reads:

$$\delta S = k_b(c - c_0)\hat{n} \tag{15}$$

where $k_b$ is a fit parameter (one per band $b$), $c$ is the color of the object in hand, $c_0$ is the average color, and $\hat{n}$ is the direction of the displacement in the tangent plane (i.e. a normalized vector along the parallactic direction, computed once for all for each Calexp). We have not accounted for pressure variations because they are usually small, but it would not be difficult. The code accounts for color-driven differential effects within a given band, but ignores the differences across bands, would one attempt to fit images from different bands at the same time. Differences in recorded positions across bands will be accounted for in the fitted mappings. It is important to do so because we are fitting WCS's, and we want the fitted mappings to reflect at best the effects affecting measured positions. Since the color correction (15) is not accounted for when using WCS's to transform measured position, we have made this correction zero on average. As for proper motions, fitting or not these refraction-induced

differential position shifts is left to the user (see Section 4.10).

## 4.8 Astrometry example

The matrix $W_{\gamma,i}$ is obtained by transporting the measurement errors through the fitted mapping. This introduces an extra dependency of the $\chi^2$ on the parameters, that we have decided not to track in the derivatives, because these errors mostly depend on the mapping scaling, which is very well determined from the beginning. However, small changes of scaling can lead to the $\chi^2$ increasing between iterations. This is why we provide the *AstrometryModel::freezeErrorScales* which allows one to use the *current* state of the model to propagate errors, even if mappings are updated. $dD_{\gamma i}/d\theta$ has two non-zero blocks: the derivatives with respect to the parameters of the $M_\gamma$ mapping (which are delivered by the Gtransfo-derived class that implements the fitted mapping, namely by the *Gtransfo::paramDerivatives* routine); and the derivative with respect to the $F_k$ position which reads $dP_\gamma(F_k)/dF_k$ (delivered as well by the class that implements the projector, via the *Gtransfo::computeDerivative* routine).

Regarding reference terms, the matrix $W_j$ should be derived from the reference catalog position uncertainty matrix $V_0$ (typically delivered for $(\alpha, \delta)$ coordinates):

$$W_j = (P'^T V_0 P')^{-1}$$

where $P'$ is the derivative of the projector. The inverse of $W_j$ is in practice obtained using the routine *Gtransfo::transformPosAndErrors* which is attached to the projector. The derivative of the reference residual $D_j$ with respect to the *FittedStar* position $F_j$ (see eq. (6)), is just the $2\times2$ matrix of the derivative $P'$ of the projector $P$, which we compute using *Gtransfo::computeDerivative*.

## 4.9 A note about our choice for linear solvers

The standard Cholesky decomposition of a matrix H consists in finding a factor $L$ such that $H = LL^T$, with L triangular (possibly after a permutation of indices). Both Eigen and Cholmod offer a variant, $H = LDL^T$, where D is diagonal and L (still triangular) has 1's on its diagonal. We have settled for this variant, because it offers improved numerical stability and allows one, if needed, to add constraints (via Lagrange multipliers) to the problem. We have also improved the Eigen interface to Cholmod by exposing to the user the factorization update capability of Cholmod, which considerably speeds up the outlier removal. This is done in the *CholmodSimplicialLDLT2* class. Using Cholmod has a drawback: we need its run-time library. Cholmod is now packaged in SuiteSparse, much bigger than what we need. This is why we

have packaged the smallest possible subset of SuiteSparse that fulfills our needs into `jointcal_cholmod`.

### 4.10 Indices of fits parameters and Fits of parameter subsets

Since we use vector algebra to represent the fit parameters, we need some sort of mechanism to associate indices in the vector parameter to some subset (e.g. the position of an FittedStar) of these parameters. Furthermore, the implementation we have chosen does not allow trivially to allocate the actual parameters at successive positions in memory. The *AstrometryFit::AssignIndices* takes care of assigning indices to all classes of parameters. For the mappings, the actual *AstrometryModel* implementation does this part of the job. All these indices are used to properly fill the Jacobian and gradient, and eventually to offset parameters in the *AstrometryFit::OffsetParams*.

Since the indexing of parameters is done dynamically, it is straightforward to only fit a subset of parameters. This is why the routine *AstrometryFit::AssignIndices* takes a string argument that specifies what is to be fitted.

## 5 Association of the input catalogs

In the LSST stack (Swinbank et al., LDM-151) framework, each reduced input image is call a "calexp" (Calibrated Exposure). Each calexp holds the data from one exposure of one CCD, and we associate the "reduction" products, typically a variance map, image mask planes, and the derived catalog and WCS obtained by matching the catalog to some external reference during single-frame processing. The data that `jointcal` needs from the calexp is stored in a *CcdImage* object. It stores the objects selected from the source catalog (using a configurable *SourceSelector*), the relevant exposure metadata and the initial WCS and *PhotoCalib*.

The *Associations* class holds the list of input *CcdImage*'s and connects together the measurements of the same object. The input measurements are called *MeasuredStar* and the common detections are called *FittedStar*. The objects collected in an external catalog are called *RefStar*. Despite their names, these classes can represent galaxies as well as stars. The collections of sush objects are stored into *MeasuredStarList*, *RefStarList* and *FittedStarList*, which are container derived from *std::list*. The relations between these classes, all implemented in C++, are displayed in Figure 2 on page 14.

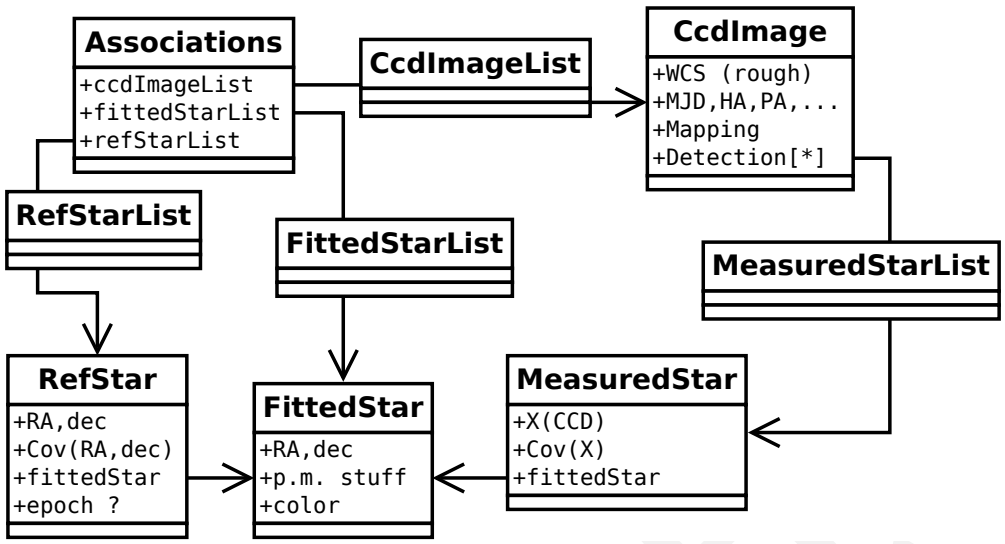


FIGURE 2: Chart of class relations which implement the associations between input catalogs. One *FittedStar* usually has several *MeasuredStar* pointing to it, and each *RefStar* points to exactly one *FittedStar*. Most *FittedStar*'s have no *RefStar*.

## 6 Fitting the transformations between a set of images

Some applications require determination of transformations between images rather than mappings on the sky. For example a simultaneous fit of PSF photometry for the computation of the light curve a point-like transient requires mappings between images to transport the common position in pixel space from some reference image to any other in the series. The calexp series would typically involve the CCD from each exposure that covers the region of interest. The package described here can fit for the needed mappings:

- in order to remove all reference terms from the $\chi^2$ of eq. (2), one avoids calling *Associations::CollectRefStars*.
- One chooses polynomial mappings for all Calexp except, reserving one to serve as a reference with a fixed identity mapping. The distortion model *SimplePolyModel* allows that.
- Chose identity projectors (the class *IdentityProjector* does precisely that).

So, fitting transformations between image sets can be done with the provided code.

# A Representation of distortions in SIP WCS's

The purpose of the appendix is to provide the minimal introduction to WCS concepts required to understand the code (and the comments) when browsing through it. Readers familiar with WCSs can give up here.

WCS's are abstract concepts meant to map data on coordinate systems. In the astronomical imaging framework, this almost always means mapping the pixel space into sidereal coordinates, expressed in some conventional space[1]. One key aspect of the WCS "system" is that it proposes some implementation of the mappings in FITS headers, which comes with software libraries to decode and encode the mappings. The WCS conventions cover a very broad scope of applications, and wide-field imaging makes use of a very small subset of those.

For the mappings used in wide-field imaging, the transformation from pixel space to sky can be pictured in two steps:

1. mapping coordinates in pixel space onto a plane.
2. de-projecting this plane to the celestial sphere.

Let us clear up the projection/de-projection step first. There are plenty of choices possible here, and the differences only matter fir really large images. The projection used by default in the imaging community seems to be the gnomonic projection: the intermediate space is a plane tangent to the celestial sphere and the plane→sphere correspondence is obtained by drawing lines that go through the center of the sphere. In practice there is no need to know that, because any software dealing with WCS's can pick up the right FITS keywords and compute the required projection and de-projection. For this gnomonic projection, one finds `CTYPE1='RA---TAN'` and `CTYPE2='DEC--TAN'` in the FITS header. This projection is often used to generate re-sampled and/or co-added images and one should keep in mind that, for large images, the pixels are not exactly iso-area. One point of convention that might be useful to keep in mind; is that WCS conventions express angles in degrees. In the gnomonic projection, offsets in the tangent plane are expressed in degrees (defined through angles along great circles at the tangent point), so that the metric in the tangent plane is ortho-normal), and sidereal angles evaluated on the sky are also provided in degrees by the standard implementations. A notable exception is the LSST software stack where, by default, the angles are provided in

[1] The WCS concepts are broad enough to accommodate mapping of planet images, but we will obviously not venture into that.

radians.

We now come back to the first mapping step, i.e. converting coordinates measured in pixel units into some intermediate coordinate system. The universal WCS convention here is pretty minimal: it allows for an affine transform, which is in general not sufficient to map the optical distortions of the imaging system, even after a clever choice of the projection. Extensions of the WCS convention have been proposed here, but none is universally understood. The LSST software stack implements the SIP addition, which consists in applying a 2-d polynomial transform to the CCD space coordinates, prior to entering the standard WCS chain (affine transform, then de-projection). In practice, the SIP "twisting" is applied by the LSST software itself (in the class *afw::image::TanWcs*), and the "standard" part (affine and de-projection, or the reverse transform) are sub-contracted to the "libwcs" code.

One common complication of the WCS arena is that it was designed in the FITS framework convention, itself highly Fortran-biased for array indexing, so that the first corner pixel of an image is indexed (1,1). The LSST software, and most modern environments use C-like indexing, i.e. images stars at (0,0), as well as coordinates in images. The WCS LSST software hides this detail to users, by offsetting the pixel space coordinates provided and obtained from the wcs-handling library.

We now detail what is involved in the SIP convention: the SIP "twisting" itself is encoded through 4 polynomials of 2 variables, which encode the direct and reverse transformations. The standard affine transform is expressed through a $2 \times 2$ matrix ($Cd$) and a reference point $X_{ref}$ (called `CRPIX` in the fits header):

$$Y_{TP} = Cd(X_{pix} - X_{ref})$$

$X_{pix}$ is a point in the CCD space, and $Y_{TP}$ is its transform in the tangent plane. Obviously, $X_{pix}$ and $X_{ref}$ should be expressed in the same frame so that the transform does not depend this frame choice. We write symbolically this transform as $Y_{TP} = L(X_{pix})$ The SIP distortions are defined by a polynomial transformation in pixel space, that we call $P_A$, for the forward transformation. By convention, the transform from pixel space to tangent plane then reads:

$$Y_{TP} = L\left(X_{pix} - X_{ref} + P_A(X_{pix} - X_{ref})\right)$$

which again does not depend on the frame choice (0-based or 1-based), provided $X_{pix}$ and $X_{ref}$ are expressed in the same frame.

In `jointcal` ,the internal representation of SIP WCS's uses three straight 2d→2d transformations: the SIP correction, the affine transformation and the de-projection. Those are just composed to yield the actual transform, and the two first ones are generic polynomial transformations. We provide routines to translate the TanWcs objects into our representation (*ConvertTanWcs*) and back (*GtransfoToTanWcs*). In the latter case, we also derive the reverse distortion polynomials, which are built if needed in our representation of SIP WCSs.

## B Notes on meas_mosaic (from HSC)

Naoki Yasuda wrote *meas_mosaic* for HSC processing, with a similar goal as `jointcal`.

For photometry, *meas_mosaic* fits a 7th order Chebyshev polynomial on the focal plane, plus a zeropoint offset per CCD. The polynomial coefficients are written to the header of **fcr-[visit]-[ccd].fits** files as **C_N_M** values, while the zeropoint and its error is written as **FLUXMAG0** and **FLUXMAG0ERR**. That calibration is applied to all of the fluxes in the catalog, which are written out to the same "*_*flux*" catalog fields (converting them to magnitudes in the process).

## References

Bernstein, G.M., Armstrong, R., Plazas, A.A., et al., 2017, PASP, 129, 074503 (arXiv:1703.01679), doi:10.1088/1538-3873/aa6c55, ADS Link

Bertin, E., 2006, In: Gabriel, C., Arviset, C., Ponz, D., Enrique, S. (eds.) Astronomical Data Analysis Software and Systems XV, vol. 351 of Astronomical Society of the Pacific Conference Series, 112, ADS Link

Eichhorn, H., 1960, Astronomische Nachrichten, 285, 233, doi:10.1002/asna.19592850507, ADS Link

Gaia Collaboration, Brown, A.G.A., Vallenari, A., et al., 2016, A&A, 595, A2 (arXiv:1609.04172), doi:10.1051/0004-6361/201629512, ADS Link

Magnier, E.A., Schlafly, E., Finkbeiner, D., et al., 2013, ApJS, 205, 20 (arXiv:1303.3634), doi:10.1088/0067-0049/205/2/20, ADS Link

Padmanabhan, N., Schlegel, D.J., Finkbeiner, D.P., et al., 2008, ApJ, 674, 1217 (arXiv:astro-ph/0703454), doi:10.1086/524677, ADS Link

**[LDM-151]**, Swinbank, J.D., et al., 2017, *Data Management Science Pipelines Design*, LDM-151, URL https://ls.st/LDM-151