Documentation of the astrometry component of the jointcal package

Pierre Astier

LPNHE/IN2P3/CNRS (Paris)

May 27, 2016

The jointcal package aims at optimizing simultaneously the WCS's of a set of astronomical images of the same field. This approach produces in principle, and often as well in practice, WCS's 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. It is meant to be used within the LSST software stack framework.

Contents

1. Introduction

With deep astronomical images, it is extremely common that the relative astrometry 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-square 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 consists mostly in two sets: the position (on the sky) of the objects in common, 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. So far, we have used the USNO (A 2.0) catalog for this purpose.

SCAMP[1] 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 somehow 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 hence is likely sub-optimal statistics wise. We do not know how sub-optimal it is, but 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.

The code is heavily biased towards astrometry. One related problem, important in some of the target applications, is relative *photometry*. We have not provided yet the determination of relative flux scales between images but we plan to do so. The principles are similar to relative astrometry, but considerably simpler. The main reason to integrate it to the present package is that loading the input catalogs takes a large amount of time and so, solving for the relative photometry once they are in memory looks like a good idea.

The plan of this note goes as follows: we first sketch the algorithm (§2). We then describe the first step, i.e. how we associate the measurements of the same objects in different exposures (§3). We provide our least-squares formulation in § 4, and describe the how we evaluate the derivatives with respect to the parameters.

2. Algorithm flow

The algorithm assumes that the input images are equipped with a WCS accurate to $\sim 1''$. Currently, the code interprets properly the SIP WCS's (relying on the IO's from afw), with or without distortions. The code might handle transparently the "PV" encoding of

[1]see http://www.astromatic.net/software/scamp

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 (the selection of objects is left to the user)
2. Associate these catalogs, i.e. associate the detections of the same object in the image set. one can also associate these ensemble of detections with an external catalog (USNO in practise).
3. Actually fit what should be fitted, with outlier clipping.
4. Output results.

3. Association of the input catalogs

In the LSST stack framework, the reduced input images are called "Calexp". Each of those typically holds the data from 1 CCD and exposure, and associate the "reduction" products, typically a variance map, a catalog and a WCS obtained by matching the catalog to some external reference. The data from a *Calexp* we need for further processing is stored into a **CcdImage** object. It basically stores informations derived from the image FITS header, from the WCS, and from the image catalog.

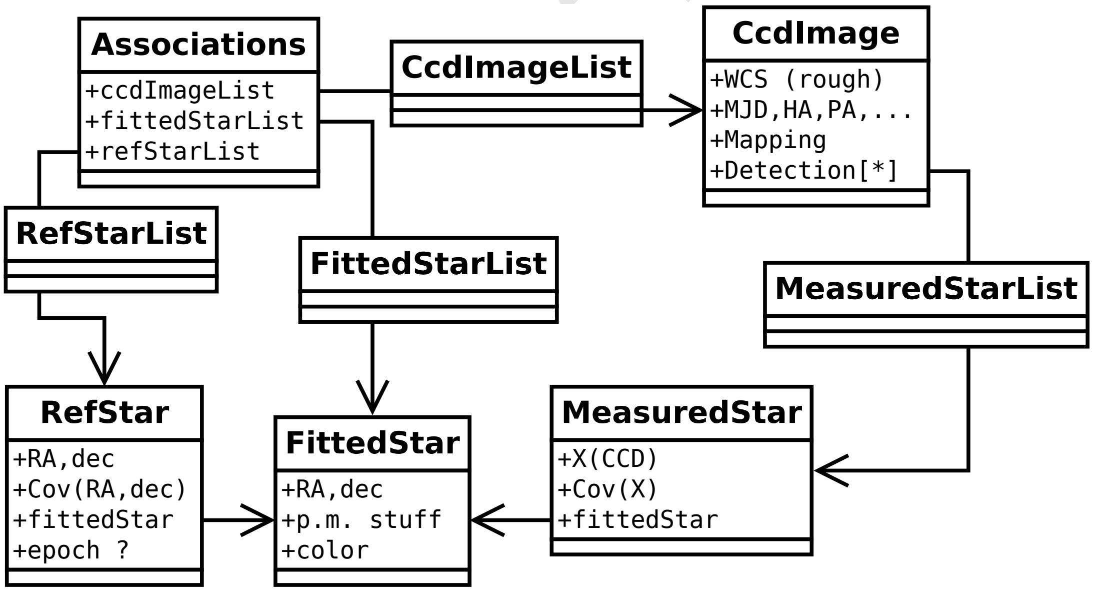

Figure 1: Chart of class relations which implement the associations between input catalogs. One **FittedStar** usually has several **MeasuredStar** pointing to it, and each **RefStar** points to exactly one **FittedStar**. Most **FittedStar's** have no **RefStar**.

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 1.

4. Least squares

4.1. Least-squares expression

The fit consists of minimizing:

$$\begin{aligned}\chi^2 = &\sum_{\gamma,i}[M_\gamma(X_{\gamma,i}) - P_\gamma(F_k)]^T W_{\gamma,i}[M_\gamma(X_{\gamma,i}) - P_\gamma(F_k)] && \text{(meas. terms)} \\ &+ \sum_j [P(F_j) - P(R_j)]^T W_j [P(F_j) - P(R_j)] && \text{(ref. terms)} \qquad (1)\end{aligned}$$

where the first line iterates on all MeasuredStar (i) from all CcdImage (indexed by γ), and the second iterates on all RefStar (j). In the first terms, the object at position F_k is the one that was measured at position $X_{\gamma,i}$ in image γ. The association between measurements and objects is described in the above paragraph.

The measurement terms compare the measurement positions to objects positions, the reference terms compare object positions to reference positions. We need these two sets of terms because not all objects F_k in the first terms appear in the second terms: there are plenty of objects in the images which are not in the reference catalogs, but which constrain the mappings M_γ to transform measured coordinates to the same positions.

For the measurement terms (first line), the notations are:

- M_γ is the mapping (for CcdImage γ) from pixel space to some tangent plane to the celestial sphere, (defined by P_γ);
- P_γ is a projector from sidereal coordinate to some tangent plane; P_γ is user-defined.
- $X_{\gamma,i}$ is the position of MeasuredStar i of CcdImage γ in pixel space of this CcdImage;
- F_k is the (sky) position of the FittedStar corresponding to this MeasuredStar. In the data structure, the FittedStar is just pointed to by the MeasuredStar see fig. 1;
- $W_{\gamma,i}$ is the measurement weight of $M_\gamma((X_{\gamma,i})$, i.e. the inverse of the 2×2 covariance matrix.

The notations for the reference terms (second line) are:

- R_j refers to the (sky) position of RefStar j
- F_j refers to the (sky) position of the corresponding FittedStar (i.e. pointed to by the RefStar, see fig. 1)
- P is some (user-provided) projector;
- W_j is the weight matrix of the projected position $P(R_j)$.

The expression 1 above depends on two sets of parameters: the parameters defining the mappings M and the positions F_k. For a practical problem, this amounts to a very large number of parameters, which becomes tractable if one remarks that every term in the $\chi 2$ only addresses a small number of parameters. We exploit this feature to compute rapidly the gradient and even the Hessian of the χ^2.

So far, we have not specified how we model the mappings M nor how we choose the various projectors that appear in expression 1. The code has been written to allow the final user to provide its own version 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.

4.2. The distortion model

The routines in the **AstromFit** 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 **DistortionModel** 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 10×36 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 of "normalizing" the input coordinates (roughly map the range of fitted data over the $[-1, 1]$ interval), in order to alleviate the well-know numerical issues associated to fitting of polynomials.

4.3. Choice of projectors

In the least squares expression 1, the residuals of the measurement terms read:

$$R_{\gamma i} = M_\gamma(X_{\gamma,i}) - P_\gamma(F_j)$$

If the coordinates F_j are sidereal coordinates, the projector P_γ determine the meaning of the mapping M_γ. If one is aiming at producing WCS's for the image, it seems wise to choose for P_γ the projection used foe the envisioned WCS, so that the mapping M_γ 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_γ can eventually be split into the "CD" matrix and the SIP-specific higher order distortion terms (see §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 **AstromFit** 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 1 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.

4.4. Proper motions and atmospheric refraction

The expression 1 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 §4.6).

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:

$$\delta X = k_b(c - c_0)\hat{n} \tag{2}$$

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 2 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 §4.6).

4.5. Minimization approach

The expression 1 depends on two sets of parameters: the parameters defining the mappings $M_\gamma(X) \equiv= M_\gamma(\eta_\gamma, X)$, and the positions F_k. There are indeed extra parameter sets and what is described here applies as well to those.

We call θ the vector that gathers all parameters. For a practical problem its size can easily reach 10^5. But the 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 η_γ parameters could be connected (in the second derivative matrix) to only a small set of F_j's. So, it is tempting to search for the χ^2 minimum using methods involving the second derivative matrix, if we take advantage of its sparseness.

Let us rewrite the χ^2 (expression 1) as:

$$\begin{aligned} \chi^2 = & \sum_{\gamma,i} R_{\gamma i}^{m\,T} W_{\gamma,i} R_{\gamma i}^{m} \\ & + \sum_j R_j^{r\,T} W_j R_j^r \qquad (3) \\ & \qquad (4) \end{aligned}$$

where the meaning or the R vectors is easily derived by comparing to 1. We want to find the point where $d\chi^2/d\theta = 0$, where θ of (size N_p) denotes the vector of parameters. We have

$$\begin{aligned} \frac{1}{2}\frac{d\chi^2}{d\theta} = & \sum_{\gamma,i} R_{\gamma i}^{m\,T} W_{\gamma,i} H_{\gamma i}^{m} \qquad (5) \\ & + \sum_j R_j^{r\,T} W_j H_j^r \qquad (6) \\ & \qquad (7) \end{aligned}$$

where the H matrices are $2 \times N_p$ in size and read:

$$\begin{aligned} H_{\gamma i}^m &= \frac{dR_{\gamma i}^m}{d\theta} \qquad (8) \\ H_j^r &= \frac{dR_j^m}{d\theta} \qquad (9) \\ & \qquad (10) \end{aligned}$$

If one is ready to evaluate the second derivative matrix of the χ^2, the standard method to zero the gradient is to Taylor expand it:

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

and solve for the offset that zeroes it to first order:

$$\delta\theta = -\left[\frac{d^2\chi^2}{d\theta^2}(\theta_0)\right]^{-1} \frac{d\chi^2}{d\theta}(\theta_0) \qquad (11)$$

At this stage, if the problem is non-linear (more precisely, if the second derivative varies rapidly) it is wise to consider a line search, i.e. to minimize $\chi^2(\theta_0 + \lambda \times \delta\theta)$ over λ.

We write the second derivative of our χ^2 (or Hessian) as:

$$\begin{aligned} \frac{1}{2}\frac{d^2\chi^2}{d\theta^2} = & \sum_{\gamma,i} H_{\gamma i}^{m\,T} W_{\gamma,i} H_{\gamma i}^{m} \\ & + \sum_j H_j^{r\,T} W_j H_j^r \qquad (12) \\ & \qquad (13) \end{aligned}$$

where we have neglected the second derivatives of the R vectors. This is both simple and handy, because this second derivative is then by construction positive-definite and hence the parameter offsets (defined in 11) can be evaluated using the (fast) Cholesky factorization. This indicates that, if possible, mappings $M_\gamma(\eta_\gamma, X)$ linear with respect to their parameter η_γ, for example polynomials, are to be favored.

Since the matrices $W_{\gamma,i}$ are positive-definite, 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 $K_{\gamma i}^m = \alpha_{\gamma i} H_{\gamma i}^m$, the Hessian expression becomes

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

The sums present in this expression can be performed using matrix algebra. We concatenate all the K matrices into a big matrix, called the Jacobian matrix:

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

and we then 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 χ^2 only depends on a small number of parameters. The data stuctures allow us to collect easily the indices of these parameters, and we evaluate in fact the H matrices for these indices only.

The computation of the Jacobian and the gradient is orchestrated in the **AstromFit** class. The routines `AstromFit::LSDerivatives1` and `AstromFit::LSDerivatives2` evaluate the contributions to the Jacobian and gradient of the χ^2 from the measurement terms and the references terms respectively. In these routines, the Jacobian is represented as a list of 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 $HX = -g$ using a Cholesky factorization. For sparse linear algebra, at least the Cholmod and Eigen packages provide the required functionalities. It turns out that for practical problems, the factorization is the most CPU intensive part 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 %.

4.6. 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 `AstromFit::AssignIndices` takes care of assigning indices to all classes of parameters. For the mappings, the actual **DistortionModel** 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 `AstromFit::OffsetParams`.

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

5. Fitting the transformations between a set of images

Some applications require to determine 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 χ^2 of eq. 1, one just avoids to call `Associations::CollectRefStars`.
- One chooses polynomial mappings for all Calexp but one which will serve as a reference and have a fixed identity mapping. The distortion model **SimplePolyModel** allows to do that.
- Chose identity projectors (the class **IdentityProjector** does precisely that).

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

6. Run example

```
        assoc = Associations()

# iterate on the input calexps
        for dataRef in ref :
            src = dataRef.get("src", immediate=True)
            calexp = dataRef.get("calexp", immediate=True)
            tanwcs = afwImage.TanWcs.cast(calexp.getWcs())
            bbox = calexp.getBBox()
            md = dataRef.get("calexp_md", immediate=True)
            calib = afwImage.Calib(md)
            filt = calexp.getFilter().getName()
# select proper sources
            newSrc = ss.select(src, calib)

# actually load the data
            assoc.AddImage(newSrc, tanwcs, md, bbox, filt, calib,
                           dataRef.dataId['visit'], dataRef.dataId['ccd'],
                           dataRef.getButler().mapper.getCameraName(),
                           astromControl)

# carry out the association
        matchCut = 3.0
        assoc.AssociateCatalogs(matchCut)
# collect reference objects
        assoc.CollectRefStars(False) # do not project RefStars
# select objects measured at least twice.
        assoc.SelectFittedStars()
# Send back fitted stars on the sky.
```

```
        assoc.DeprojectFittedStars() # required for AstromFit
# Chose a ProjectionHandler
        sky2TP = OneTPPerShoot(assoc.TheCcdImageList())
# chose a distortion model
        spm = SimplePolyModel(assoc.TheCcdImageList(), sky2TP, True, 0)
# Assemble the whole thing
        fit = AstromFit(assoc,spm)
# we are ready to run. Initialize parameters by running "partial fits"
        fit.Minimize("Distortions")
        fit.Minimize("Positions")
# now fit both sets simultaneously.
        fit.Minimize("Distortions Positions")
# output residual tuples
        fit.MakeResTuple("res.list")
```

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[2]. 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$\rightarrow$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

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

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 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 2×2 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$\rightarrow$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.