The i2k Align and i2k Align Retina Toolkits: Correspondences and Transformations

Charles V. Stewart
DualAlign LLC
Clifton Park, NY 12065
chuck@dualalign.com

June 30, 2009

This document describes the output format for the correspondences and the transformations from the *i2k Align* and *i2k Align Retina* libraries. These transformations coincide with the layout options available the software, thus by reading through this document users who are not particularly interested in the output can gain an understanding of the effect of the transformations.

The functionality of *i2k Align Retina* is a superset of the functionality of *i2k Align*, but also includes tools for aligning and montaging retinal images. For the sake of simplicity, this document will use the name *i2k Align* generically to describe both libraries as though they are a single library and only refer to *i2k Align Retina* for clarification as needed.

Users can access either library through the UI (see on-line documentation accessible when launching the UI), through a command-line interface (see separate document), or by linking their own program against the libary (see `da_i2k_align.h`).

While the library's primary output is images — a single image by the montage tool or one file per aligned image by the align tool — the user has the choice to have the correspondences and the transformations saved as well. The correspondences and transformations are saved as text files in the same output folder as the image(s). The files names have the same prefix as the images, and are appended with the strings ¨`_correspondences.txt`¨ and ¨`_xforms.txt`¨.

Coordinate Systems

Locations in the images handled by *i2k Align* are described in a coordinate system where the upper left corner is the origin — with $(0,0)^\top$ being the center of the upper left pixel. As usual, x values increase across the image, and y values increase down the image, and the units of the coordinate system correspond to the pixel dimensions at the input image resolution. Thus, each increment of 1 in an x or y value corresponds to a change of 1 pixel across or down in the original images. All correspondences are described in these units.

Each image is transformed onto a coordinate system that will be referred to here as the *aligned* coordinate system. Each transformed image does not necessarily have its upper left coordinate at the $(0,0)^\top$ point of this coordinate system. Instead, there is a global offset $(u_0, v_0)^\top$. By subtracting this offset one obtains the final coordinates in the output images, which we refer to as the *montage* coordinate system. Thus, if $(u,v)^\top$ is a point the aligned coordinate system, then $(u-u_0, v-v_0)^\top$ is the pixel location in the montage coordinate system.

Transformations, Layouts and The Transformation File

Seven different transformations can be output by *i2k Align*: translation, similarity, affine, homography, homography plus radial lens, cylindrical, and quadratic (*i2k Align Retina* only). The different types of transformations are created in response to the layout options of *i2k Align*. In particular, translation is the Reposition Layout, homography plus radial lens is the Planar Layout, and cylindrical is the Cylindrical Layout. These three are used for montaging photographic images, with the Auto Layout option automatically choosing between them. For non-photographic images and for aligning photographic images, the similarity, affine and homography transformations correspond to the layout options of the same name, and the planar and cylindrical layout / transformations are reused, giving five options. Note that similarity, affine, homography and planar are "nested", meaning that each is a restricted case of the next. Quadratic transformations are used for retinal images and may not be chosen explicitly.

The format of the output of each transformation is described below. Before the transformations are output, there are three initial lines of output:

```
NUMBER_OF_IMAGES n
MONTAGE_ORIGIN u_0 v_0
ANCHOR_IMAGE_NAME file-name
```

where `n` ($n \geq 2$) is an integer giving the number of aligned images, which of course is also the number of output transformations. (Note: if *i2k Align* discovers that there are two or more disconnected groups of images, then it will only output the transformations associated with the largest group.) The values `u_0` and `v_0` are the output of $(u_0, v_0)^\top$ described above. The anchor image, `file-name` is the image (original name) that is roughly in the center of the montaged / aligned set. If no final distortion correction is applied, then the transformation associated with the anchor image will be quite simple. On the other hand, if distortion correction is applied, then this image's transformation will be just as a complicated as any other.

The name of `file-name` will have two different formats, depending on whether the command-line executable is used or the UI is used. For the command-line executable, `file-name` will be exactly as it appears in the input file list. (In other words, one image from the input file list will have been chosen automatically by *i2k Align* as the anchor.) For the UI, `file-name` will be the full-path to the input file (this is currently subject to change). In either case, some care is needed in handling of the file names: they can not be treated directly as strings in a programming language because they may have blank characters in the middle of them. This same observation applies to all file names in the transformations file and in the correspondences file.

Following these three lines of output, each of the n transformations will be output. Each output will start with the file-name, just as described above, followed by the name of the transformation type, followed by the transformation itself. The transformations and names are described in the following sections. In reading the descriptions, remember that these are transformations onto the *aligned* coordinate system and that $(u_0, v_0)^\top$ must be subtracted to obtained the value in the *montage* coordinate system of the final output image(s).

Translation

For translation, after the file name there are two more lines of output:

```
TRANSLATION
t_x t_y
```

where `t_x` (t_x) and `t_y` (t_y) are double-precision translation values. Thus, the mapping of input image location $(x, y)^\top$ onto the *aligned* coordinate system is

$$u = x + t_x$$
$$v = y + t_y$$

As described above, translation corresponds to the `Reposition` layout option, and only applies to photographic images.

Similarity Transformation and Layout

For the similarity transformation, the output (after the file name) is

```
SIMILARITY
a_00 a_01 t_x
a_10 a_11 t_y
```

where `a_ij` (a_{ij}) and `t_x` (t_x) and `t_y` (t_y) are double-precision values. The mapping of image location $(x, y)^\top$ onto the *aligned* coordinate system is

$$u = a_{00}x + a_{01}y + t_x$$
$$v = a_{10}x + a_{11}y + t_y$$

Because this is a similarity transformation, $a_{00} = a_{11}$ and $a_{10} = -a_{01}$. Thus, the output is redundant. It is written this way for convenience in describing the effect of the transformation.

Affine Transformation and Layout

For the affine transformation the output is

```
AFFINE
a_00 a_01 t_x
a_10 a_11 t_y
```

where `a_ij` (a_{ij}) and `t_x` (t_x) and `t_y` (t_y) are double-precision values. The mapping of $(x, y)^\top$ onto the *aligned* coordinate system is

$$u = a_{00}x + a_{01}y + t_x$$
$$v = a_{10}x + a_{11}y + t_y$$

While the parameters and the transformation have the same form as the similarity transformation, the restrictions on the a_{ij} values do not apply. The only restriction is that $a_{00}a_{11} - a_{10}a_{01} \neq 0$, so there are six degrees of freedom to the transformation.

Homography Transformation and Layout

For the homography transformation, the output is

```
HOMOGRAPHY
h_00 h_01 h_02
h_10 h_11 h_12
h_20 h_21 h_22
```

where the `h_ij` (h_{ij}) values are double-precision. These nine values can be seen as the parameters of a planar projective transformation or homography. The 3x3 homography matrix **H** formed by these values is invertible. The mapping of a point onto the *aligned* image coordinate system is easily described in two steps:

$$u' = h_{00}x + h_{01}y + h_{02}$$
$$v' = h_{10}x + h_{11}y + h_{12}$$
$$w' = h_{20}x + h_{21}y + h_{22}$$

and

$$u = u'/w'$$
$$v = v'/w'$$

In theory, w' could be 0, but in practice this does not happen for a transformation that *i2k Align* accepts as correct. Moreover, usually, h_{20} and h_{21} will be small and h_{22} will be at or close to 1.

This transformation, like the previous four, is invertible, so the mapping from the *aligned* coordinate system back to the original coordinate system is straightforward.

Homography with Radial Lens Transformation / Planar Layout

The homography-with-radial-lens transformation, corresponding to the Planar Layout, is the same as the homography, with the addition of a radial lens distortion term. This is called "Planar" because it captures realistically and accurately the transformation of images taken of a flat surface, with the homography component capturing change in viewpoint and the radial lens term capturing camera distortions. In theory, the radial lens term should be the same for all images, but in practice we have found it easier and more accurate to have a separate term for each.[1]

The transformation itself is described with the nine homography terms plus four additional terms

```
HOMOGRAPHY_WITH_RADIAL
h_00 h_01 h_02
h_10 h_11 h_12
h_20 h_21 h_22
k_1 k_2 x_c y_c
```

where all values are double-precision. Of particular note, the value k_1 is the radial lens term, the value k_2 is the (approximate) inverse radial lens term, and x_c and y_c are the center of the image.

The application of the transformation involves first applying the radial lens term, then mapping onto the *aligned* coordinate system. Starting again with $(x, y)^\top$ in the image coordinate system (uncentered!), compute

$$x' = x_c + (x - x_c)(1 + k_1 r^2)$$
$$y' = y_c + (y - y_c)(1 + k_1 r^2)$$

[1] Users of the GDB-ICP software will recall that there were two radial lens terms used there, but that was for an alignment of two images. Here, if two images are aligned (and no distortion correction is applied), the anchor image transformation will be the identity plus one radial lens term, giving the tenth degree of freedom.

where $r^2 = (x - x_c)^2 + (y - y_c)^2$. Using this, we can apply the homgraphy transformation as before:

$$u' = h_{00}x' + h_{01}y' + h_{02}$$
$$v' = h_{10}x' + h_{11}y' + h_{12}$$
$$w' = h_{20}x' + h_{21}y' + h_{22}$$

and

$$u = u'/w'$$
$$v = v'/w'$$

The transformation can be approximately inverted. Mapping from $(u, v)^\top$ in the aligned coordinate system to $(x', y')^\top$ involves applying the inverse of the **H** homography matrix. Going from $(x', y')^\top$ to $(x, y)^\top$ in the original image coordinate system requires using the inverse radial lens term:

$$x = x_c + (x' - x_c)(1 + k_2 r'^2)$$
$$y = y_c + (y' - y_c)(1 + k_2 r'^2)$$

where $r'^2 = (x' - x_c)^2 + (y' - y_c)^2$. This does not produce an exact inverse, but in most cases it is sufficiently close.

Cylindrical Transformation and Layout

The cylindrical transformation / layout starts the same as the homography-with-radial-lens transformation, but then maps onto a sphere (not a cylinder, despite the name). By unwrapping the sphere onto a plane, the final pixel values in the *aligned* coordinate system are obtained. The additional complication is that the transformation also allows for wrap-around in θ (longitude), but not in ϕ (lattitude) dimension. These details are explained below.

The output involves three more double-precision parameters

```
CYLINDRICAL
h_00 h_01 h_02
h_10 h_11 h_12
h_20 h_21 h_22
k_1 k_2 x_c y_c
R neg pos
```

R is the sphere radius, while neg and pos are offsets used for wrap-around in θ values. R is the same across all transformations in the file, but R is repeated for each image in the output for the sake of convenience.

The mapping proceeds as above until the $(u', v', w')^\top$ values are obtained. The next step is to convert to spherical coordinate angles:

$$\theta = \text{atan2}(u', Rw')$$
$$\phi = \sin^{-1}\left[v'/\sqrt{u'^2 + v'^2 + (Rw')^2}\right]$$

Before converting these to aligned coordinates, an offset is added to θ, depending on its sign. In particular, if $\theta < 0$ then neg is *added* to θ. Otherwise, pos is added to θ. Until the sweep of the camera begins to approach a half-circle or more, the values of neg and pos will be 0. Finally, the angles are converted to aligned pixel values simply as

$$u = R\theta$$
$$v = R\phi$$

The primary step in inverting this transformation is solving for (u', v', w') from θ, ϕ and R. In this step, neg and pos do not need to be considered. Then the same process as inverting the homography-with-radial-lens transformation can be applied.

Quadratic Transformation and Layout

The quadratic transformation, available only with *i2k Align Retina*, uses twelve degrees of freedom and x^2, xy and y^2 terms. In the file, the transformation is described as

```
QUADRATIC
q_00 q_01 q_02 q_03 q_04 q_05
q_10 q_11 q_12 q_13 q_14 q_15
```

and the mapping of $(x, y)^\top$ is

$$u = q_{00}x^2 + q_{01}y^2 + q_{02}xy + q_{03}x + q_{04}y + q_{05}$$
$$v = q_{10}x^2 + q_{11}y^2 + q_{12}xy + q_{13}x + q_{14}y + q_{15}$$

Unfortunately, this transformation is not directly invertible, and so any mapping must be inverted numerically.

Correspondences

The correspondences (or "matches") file is also a text file, with a very simple format. It starts with the line

```
NUMBER_OF_MATCH_SETS p
```

where p is the number of image pairs that produced inter-image matches. If there are n aligned images, then $p \geq n - 1$. The case $p = n - 1$ generally occurs if the images are taken in a single sweep of the camera with less than 50% overlap between adjacent pairs.

There is a blank line before each match set.

Each match set starts with three lines:

```
FROM1_IMAGE_NAME file-name-1
FROM2_IMAGE_NAME file-name-2
NUMBER_OF_MATCHES m
```

where `file-name-1` gives the name of the first image, `file-name-2` gives the name of the second image, and `m` (m) is an integer giving the number of correspondences. (Again, users should be aware that there may be blanks in the file of the file names.)

There are then m lines, with each line specifying a correspondence.

Each correspondence is specified by 17 double-precision values(!). The first value is the weight assigned to the correspondence, with low weight implying that *i2k Align* gave low influence to this correspondence. Then, there are 8 values for the matching point from the first image (associated with `file-name-1`) and 8 values for its corresponding matching point form the second image (associated with `file-name-2`). In each case, these 8 values are the following 4 two-component vectors:

- $\mathbf{x}_i$: the edge element location in image i;
- $\boldsymbol{\eta}_i$: the unit vector normal to the edge element in image i;
- $\mathbf{x}'_i$: the location of the edge element from image i after it has been mapped to the *aligned* coordinate system;
- $\mathbf{y}_i$: the location of a "pseudo-corner" constructed at the point.

These require some explanation.

The core registration algorithm of *i2k Align* is based on point-to-line constraints, both for registering two images and for ultimately computing the final transformations onto the *aligned* coordinate system. These constraints require both locations and normal directions. To illustrate, if we were trying to compute, say, the affine transformation of image 1 onto image 2, estimating the 2x2 matrix $\mathbf{A}$ and the translation vector $\mathbf{t}$, then the squared point-to-line error of a correspondence is

$$e_{12}^2 = w_{12}\big[(\mathbf{A}\mathbf{x}_1 + \mathbf{t} - \mathbf{x}_2)^\top \boldsymbol{\eta}_2\big]^2,$$

where w_{ij} is the weight (the first value on the input line for this correspondence). Summing over all correspondences for image 1 and 2 gives a weighted least-squares objective function. Minimizing this objective function with respect to the six parameters of $\mathbf{A}$ and $\mathbf{t}$ gives a good estimate of the inter-image transformation. This estimate will be better than the one obtained using point-to-point distances, with squared error

$$e_{12}^2 = w_{12}\|\mathbf{A}\mathbf{x}_1 + \mathbf{t} - \mathbf{x}_2\|^2.$$

The third vector, $\mathbf{x}'$, for each point involved in a correspondence (on the line of input, values 6 and 7 for the image 1 point and 14 and 15 for the image 2 point) is the point that is obtained by applying the transformation onto the *aligned* coordinate system. Therefore this can be used to verify the correctness of a transformation.

The fourth and final vector for each point, $\mathbf{y}_i$, is an artificial corner point created for users who need point-to-point correspondences, primarily for purposes other than computing inter-image transformations. The vectors $\mathbf{y}_1$ and $\mathbf{y}_2$ (from image 1 and image 2) are constructed to be closer to each other (after application of the transformations!) than $\mathbf{x}'_1$ and $\mathbf{x}'_2$ are, while still respecting the point-to-line constraints and other constraints that arise from the details of forming the edges and the correspondences originally. While these details might be unclear, the important point to remember is that if point-to-point constraints are needed, $\mathbf{y}_1$ and $\mathbf{y}_2$ should be used instead of $\mathbf{x}_1$ and $\mathbf{x}_2$. The user should also remember to always use the weight value w_{12}.