AUTOMATIC REGISTRATION OF MULTISPECTRAL IMAGES

By

BRENDAN TOBIN

A Thesis Submitted to The Honors College

In Partial Fulfillment of the Bachelors Degree
With Honors in

Materials Science and Engineering

THE UNIVERSITY OF ARIZONA

MAY 2013

Approved by:

Dr. Robert Erdmann
Department of Materials Science and Engineering

The University of Arizona Electronic Theses and Dissertations
Reproduction and Distribution Rights Form

The UA Campus Repository supports the dissemination and preservation of scholarship produced by University of Arizona faculty, researchers, and students. The University Library, in collaboration with the Honors College, has established a collection in the UA Campus Repository to share, archive, and preserve undergraduate Honors theses.

Theses that are submitted to the UA Campus Repository are available for public view. Submission of your thesis to the Repository provides an opportunity for you to showcase your work to graduate schools and future employers. It also allows for your work to be accessed by others in your discipline, enabling you to contribute to the knowledge base in your field. Your signature on this consent form will determine whether your thesis is included in the repository.

Name (Last, First, Middle)
Tobin, Brendan, L

Degree title (eg BA, BS, BSE, BSB, BFA):
BS

Honors area (eg Molecular and Cellular Biology, English, Studio Art):
MATERIALS SCIENCE AND ENGINEERING

Date thesis submitted to Honors College:
MAY 2013

Title of Honors thesis:
AUTOMATIC REGISTRATION OF MULTISPECTRAL IMAGES

The University of Arizona Library Release Agreement

I hereby grant to the University of Arizona Library the nonexclusive worldwide right to reproduce and distribute my dissertation or thesis and abstract (herein, the "licensed materials"), in whole or in part, in any and all media of distribution and in any format in existence now or developed in the future. I represent and warrant to the University of Arizona that the licensed materials are my original work, that I am the sole owner of all rights in and to the licensed materials, and that none of the licensed materials infringe or violate the rights of others. I further represent that I have obtained all necessary rights to permit the University of Arizona Library to reproduce and distribute any nonpublic third party software necessary to access, display, run or print my dissertation or thesis. I acknowledge that University of Arizona Library may elect not to distribute my dissertation or thesis in digital format if, in its reasonable judgment, it believes all such rights have not been secured.

☒ Yes, make my thesis available in the UA Campus Repository!

Student signature: [signature] Date: 5/6/13

Thesis advisor signature: [signature] Date: 5/6/13

☐ No, do not release my thesis to the UA Campus Repository.

Student signature: _____ Date: _____

Last updated: 04/01/13

Abstract

Multispectral imaging is an important tool in art conservation because it allows researchers to view under-drawing and restoration efforts on a painting. It is necessary to precisely align various images of the same painting using digital image processing techniques, however this is complicated by inconsistencies in image contents and the use of multiple cameras. This paper seeks to develop an effective method to automatically align and rescale multispectral images of paintings. Key point identification using the Speeded-Up Robust Feature detection algorithm with refinement by Random Sample Consensus fitting was determined to be an accurate and efficient method for aligning visible and infrared multispectral images.

Introduction

Different substances are transparent to different wavelengths of light. This phenomena is the reason an individual does not get a sunburn when sitting behind a window. The ultraviolet light responsible for a sunburn is mostly absorbed by the glass, while visible light travels through the glass uninterrupted. In the field of art conservation this property can be exploited to ‘see’ what is beneath a layer of paint using the correct wavelengths of light. Oil paints are typically transparent to infrared light, while graphite and chalk, which is commonly used by artists to sketch on a canvas, absorbs infrared light. Similarly most organic materials in a painting are transparent to x-rays, while metals, wood, and some modern pigments absorb x-ray light. Art conservators use a combination of infrared, visible, raking light, and x-ray photography to assemble a profile of images of a painting, which is an invaluable tool when studying the history of a painting. Figure 1 shows a typical image profile for a painting, which reveals that a figure that has been painted over. A multispectral image profile can reveal under-drawings, modifications, previous restorations, and structural elements, all of which are important to the history and conservation of a painting.

Figure 1. Hieronymus Bosch. *St. John the Baptist in the Wilderness*. Oil. C. 1489. Museo Lázaro Galdiano.
(Top Right) Visible spectrum image of the painting.
(Top Left) Infrared Reflectogram showing the portrait of a donor which has later been painted over. The dark pigments used for the animals in the background have a much stronger absorbance in the infrared than the surrounding pigments.
(Bottom Left) X-ray radiograph of the painting showing the wooden cross braces supporting the back of the painting. Both the face of the donor and the fruit obscuring his head are visible. The white pigments used in St. John's robe and the lamb have a higher reflectance to x-rays than the surrounding pigments.

In order to most effectively extract information from a multispectral image profile, it is critical to *register* the images, or precisely align and overlay all of the images in a series. This makes it possible to tell exactly where underlying features occur, which is important both in restoration efforts and analyzing an artist's intentions and design choices. During restoration and cleaning x-ray images can show where previous damage has occurred, as well as how it has been repaired. Skilled restorations can be virtually invisible to the naked eye, despite potentially concealing significant damage to a painting. Understanding the exact location of damage and restoration can assist conservators in cleaning or making further repairs to a painting. *(*Fontana*,* 2007*)* The infrared signatures of paint pigments can also help detect and identify previous restorations. In some practices of conservation it was common to repaint damaged or aging works in order to repair chips or return paintings closer to their 'original' appearance. Unfortunately some restorations made use of modern paints which contain metal based pigments and alternate binding media. When cleaning or treating a painting it is very important to know the chemical composition of the paints present because different types of paint are susceptible to different solvents. Restorations or repainting with incompatible paints can complicate cleaning if the alternate pigments are not easily discernible. Radically different types of paint can appear very similar to the naked eye, but appear quite different in an infrared image. This is because modern paints often make use of metal pigments which appear quite opaque or reflective to infrared light, and stand in sharp contrast to primarily organic oil paints. (Fontana**,** 2007) If a painting does contain incompatible pigment types, a combination of properly registered infrared

and x-ray images precisely mapped to visible images can assist in cleaning around sensitive areas.

Improvements in digital photography and cameras have made it much easier to create high resolution image profiles. Unfortunately, different cameras and conditions are necessary to acquire each image in a multispectral profile. Unless all images are taken from exactly the same vantage point and at the same magnification and resolution, there will be differences between the images in a series. Special infrared lenses and cameras are necessary to acquire infrared images, and an x-ray source and detector are required for x-ray radiographs. Additionally, it is common to image large paintings in many close-up sections in order to achieve extremely high resolution. The need for different detectors and imaging conditions makes it difficult to effectively align and replicate the same images in multiple spectrums. It is impossible for photographers to image a painting using several different cameras and light sources with the precision necessary to align a set of images.

Registration Methods

Several methods have been developed to align multispectral images. Either before or after an image series is acquired, it is necessary to identify common reference points in those images, and either physically or digitally align the images so they all appear to be taken from the same vantage point. In both cases, physical and digital, a high degree of automation is desired. A technique should not heavily rely on researchers to perform time consuming tasks such as carefully aligning a camera setup, or manually identifying reference points in a large series of images.

Scanner Mounted Spectrometer

One solution to the problem of registering multispectral images is a physical apparatus which ensures each image is captured from the same orientation. It would be conceivable to develop a frame which different cameras could be mounted on and translated to the same points for imaging. Ideally a frame would be able to effectively accommodate paintings ranging from 1/2 m to several meters across. While possible, the size and complexity of the frame would make it time consuming to construct and align effectively. Furthermore, it would probably be difficult to move, meaning paintings would have to be transported to a frame to be imaged. Moving paintings, even within the same museum, is not an easy matter, and transporting paintings long distances requires specially designed, climate controlled shipping containers. It is not viable to transport an entire collection to be studied to a research facility.

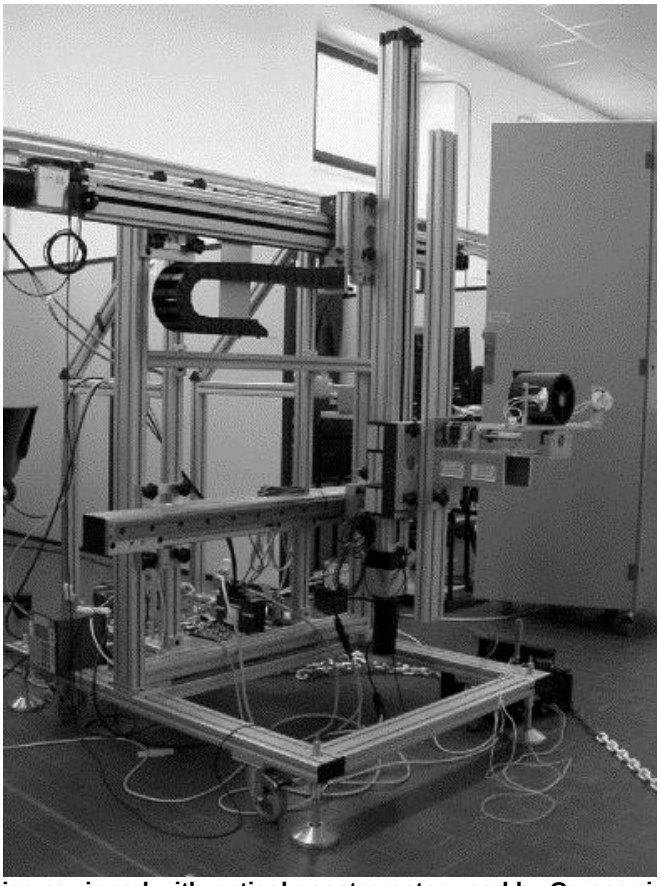


Figure 2. Scanning device equipped with optical spectrometer used by Carcagni et al. (Carcagnia, 2007)

Carcagni et al. has developed a scanner setup in which a spectrometer is scanned across a painting, registering a complete spectrum of reflectance at each point. (Carcagnia, 2007) This produces a complete set of monochromatic images, which are perfectly aligned with each other. The complete spectral response of a pigment can assist in identifying and comparing pigments, and is especially important when studying and comparing media. However, complete spectral response data is less useful when studying the content and under-drawing of a painting. Unfortunately this technique requires a large frame, spectrometer, and controlled light source which make it difficult to transport. Furthermore, this method would likely produce lower resolution images than close photographs with a high quality digital camera. While it provides superior

data regarding spectral response, the specialized equipment and need to transport paintings make this method impractical for widespread use.

Digital Assembly

A more common method for acquiring a multispectral image profile involves using digital image processing techniques to stitch close proximity sections of an image into a complete unbroken image of an entire painting. Sophisticated mosaic algorithms exist to stitch constituent images into a complete image, as well as adjust brightness to be consistent across the image and to obscure lines generated from the edge of images. Mosaic stitching techniques are well established for use in a variety of consumer and scientific applications. Provided all constituent images are taken at roughly the same angle and distance from the image plane, the results of stitching should be quite accurate.

While methodology for assembling images is well established, precisely aligning separate images from different cameras and lighting conditions presents a more difficult problem. As evident in Figure 1, the content of images acquired in various spectrums will be similar--but not identical--to that of a visible spectrum image of the same painting, and have the potential to contain under-drawings and other features not present in the visible image. The presence of additional features and absence of others complicates the process of aligning images.

This work seeks to develop and refine a robust technique to automatically register and compare multispectral images using digital image processing techniques for use in the Hieronymus Bosch Research and Conservation Project (BRCP). A

registration algorithm must be able to correct for all of the geometric distortions which may be present in an image due to camera misalignment and be accurate to within one pixel while performing as efficiently as possible. After registration visualization techniques should facilitate effective comparison of multispectral images for a human viewer.

Methodology

In order to align a pair of images it is necessary to determine the proper horizontal and vertical placement of the images relative to one another. If the two images were taken at different angles from one another, it is necessary to transform one image to appear as though it were taken from a different angle. For example a square viewed at an angle appears ‘stretched’ and looks like a diamond, but this view can be skewed and made square again. Digital techniques for modifying an image in this manner are well established, but require human input to determine the correct modification. In order to automatically determine the correct orientation and camera angle modification to align two images, corresponding points in both images may be automatically selected and used to calculate the correct distortion to the image. The process of alignment by key points involves the following steps. Identification of key points in both images, determine which key points correspond between the two images, refine the key point set to select the best matching pairs, and transform the image based upon the identified key point set.

Camera-Angle Modification

Feature point matching is a powerful tool for image registration in a variety of applications. Feature point-based registration is fast because it minimally relies on iterative techniques. The basic steps in feature point-based registration are: identification of key points in two images to be compared, matching and refinement of key point sets, and applying a geometric transformation to align the images. (Goshtasby, 2005) Camera perspective transforms are well understood in image processing. Given a set of three corresponding points from each of the images to be aligned, a homography transform can be calculated which when applied will modify the appearance of an image to accounts for all of the ways in which camera angle can affect the appearance. The geometric parameters affected are: vertical and horizontal translation, scale, aspect ratio, vertical and horizontal perspective tilt, rotation, and angle difference. Together these parameters can be summarized as a 3 by 3 transformation homography matrix (M). When the perspective matrix is applied to an image according to the equation: $dst(x,y) = \frac{(M_{11}x+M_{12}y+M_{13})}{(M_{31}x+M_{31}y+M_{33})}, \frac{(M_{21}x+M_{22}y+M_{23})}{(M_{31}x+M_{31}y+M_{33})}$ the matrix will alter the image to appear as though captured by a camera at a different location. (Bradski, 2000)

In order to register a pair of images, the set of 3 points which are known to be equivalent must first be determined. It is fairly easy for a human to identify corresponding points in an image, such as the facial features of subjects or corners of other objects in the scene. However, when dealing with large sets of images it becomes time-consuming and impractical for a researcher to manually identify matching points. Automated feature point recognition can instead be used to identify critical points in a pair of images.

Key Point Generation

Feature point recognition is a technique used in a variety of image processing fields including facial recognition and handwriting analysis. Feature point recognition involves using a square mapping technique capable of detecting similar forms regardless of their relative size and orientation in an image. In a detailed image key points will be features such as the corners of large forms, facial details, and other significant points in the image. The foundational technique, Scale Invariant Feature Transform (SIFT) was developed by David Lowe at the University of British Columbia. (Lowe, 2004) A faster, more robust technique SURF (Speeded Up Robust Features) was later developed by Herbert Bay et al. and is currently one of the preferred feature point detection algorithms. SURF generates a 128 dimension descriptor vector which characterizes the shape of a key point and can be used to compare the shape of two features. (Bay, 2006) There are a variety of implementations of SIFT and SURF in common image processing packages including OpenCV and the Python Imaging Library. OpenCV is useful because it utilizes standard NumPy array data structures for images, making it highly compatible with other packages in Python. A SURF feature detector can be used to effectively generate a set of key points for both images to be registered.

In order identify a matching set of key points for use in alignment, it is important to identify a sufficient number of potential key points, several thousand is ideal. A large number of identified key points ensures that a sufficient number of corresponding key points will be included in both sets. The maximum number of features which may be identified is limited by the size of the image and the amount of detail present in the

image. The OpenCV implementation of SURF identifies several thousand feature points for a typical 1000 x 1000 pixel image, but the number falls off quickly for images smaller than 1000 x 1000. For this reason 1000 x 1000, with a moderate level of detail appears to be the smallest size which can be reliably identified by this method.

Determining Key Point Matches

After identifying a set of key points in both images it is necessary to compare those key points and identify a set of corresponding matches. A Nearest Neighbor Search (NNS) algorithm is an effective tool for determining corresponding points. It is an optimization which will find the Euclidian distance between points, and can be applied to vectors of any length. A nearest neighbor search applied to the sets of 128-element descriptors from two images identifies the most similar key points in the two images. The ‘proximity’ of key points in 128 dimensional space corresponds to their similarity, so that the most similar key points are nearest neighbors. After performing an NNS it is necessary to further refine the set by removing redundant key point matches. It is possible for a point to be the nearest neighbor to multiple other points, though it will only have one nearest neighbor itself. Because a unique set of key point matches is desired, it is necessary to remove extra matched pairs from the set. For redundant points, the Euclidian distance between each pair is considered, and the closest pair is preserved, while the other matched pairs are removed from the set, leaving only one corresponding match for each point in the sets.

Figure 3. Key points identified after a distance threshold of 0.4 is applied. Key points from the visible spectrum image are shown as red circles and infrared spectrum image as yellow triangles overlaid on the visible spectrum image. 15 out of 20 points are true matches.

A distance threshold may be applied to refine the set of key points and select only the closest matches. Furthermore, if image size is reduced below 1000 x 1000 pixels insufficient matches may be generated. Refining the set of matches improves the speed of the RANSAC algorithm in the next step substantially. A set of at least 20 matching key points, with 75% being true matches is necessary to reliably produce a correct homography. Between 20 and 100 point pairs is sufficient to guarantee a reasonable distribution of points around the image. If too many points are clustered in a small are of the image the resultant homography will be less reliable.

Random Sample Consensus Fitting

After similar key points are identified, there is no guarantee each will be a true match. The key points identified in each image will not identify the same set of points, and as of such not every nearest neighbor match will correctly identify a set matching key points. It is necessary to filter the dataset in order to eliminate false matches and select a reasonably accurate set of matches to base image rescaling on. The Random Sample Consensus algorithm RANSAC is effective for identifying a reliable set of key point matches. (Fischer, 1981) RANSAC is an iterative method used to identify outliers in a set of data with a mathematical model. In this case the data model assumes that a specific Perspective Transform applied to one key point set will bring each key point into alignment with its match from the other key point set. Incorrectly identified match pairs will be outliers to the dataset because no perspective transform will bring those points and most of the dataset into alignment. The RANSAC algorithm iterates through the process of selecting values for the predicted model (in this case the necessary perspective transform) and then selects which points are inliers to the current model. After selecting which values are inliers, the model is recalculated using the new set of inliers and the accuracy of the model is determined from the error of the data points. The process of selecting inliers is repeated and a new model is calculated again for each iteration. Sufficient iterations should result in a very accurate estimate for a reasonably correlated dataset.

Image size may be reduced prior to selecting key points and calculating a homography in order to improve the speed of the process. No perceptible loss in

accuracy is lost when image dimensions are reduced by half, but further reductions in size result in a loss of accuracy.

Because RANSAC is a random iterative technique it will produce marginally different results each time it is applied. With a sufficient number of key points the variability of results is below pixel resolution however. The horizontal and vertical displacement from the resultant transformation homography (plots 4 and 5 in figure 4) have a standard deviation of 0.25 pixels, meaning the resultant transformation will yield consistent results. However if the translational shifts are multiplied by a factor greater than 2 in order to compensate for reductions in image size, single pixel accuracy cannot be guaranteed.

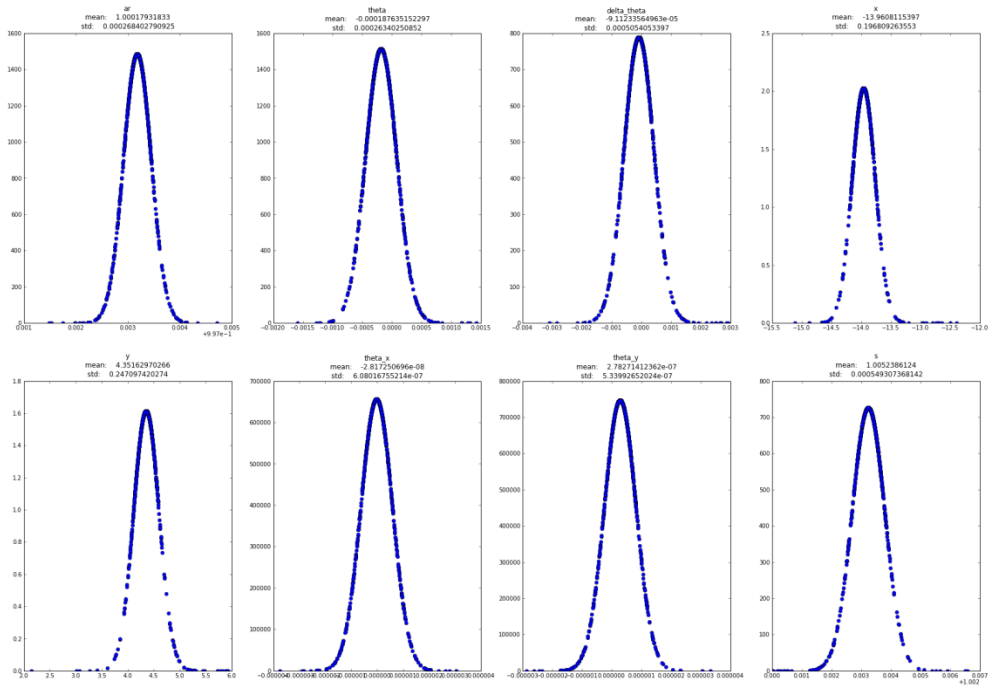


Figure 4. Histograms of the geometric parameters for the perspective transform produced from 500 iterations of registration applied to the same image. X and Y (plots 4 and 5) are horizontal and vertical displacement values.

Visualization

False color composite images are a convenient way to visualize the overlay of data from multiple images, and are commonly used in satellite sensing which makes use of visible and infrared imaging (Liew, 2001). A false color image is created by assigning full gray scale images to the different color channels of an image. In the example below an infrared spectrum image is assigned to the green channel, and a visible image is assigned to the red channel, producing the resulting images. Content only present in the visible image will appear green, content only present in the infrared image will appear red, and content present in both appears yellow, from the combination of red and green. It seems as though red and green are the most distinguishable colors to use when comparing images. Red and green are opposites in traditional color theory, and the combination yellow is quite distinguishable from both red and green. Red and blue, and blue and green are both closer to each other on a traditional color wheel, and the colors made up by their combinations seem less distinguishable to the human eye. Areas where the same form is present, but there is a significantly stronger spectral response in one imaging mode, will appear uniformly more red or green, however the consistent hue and uniform edges should make it obvious when this is the case.

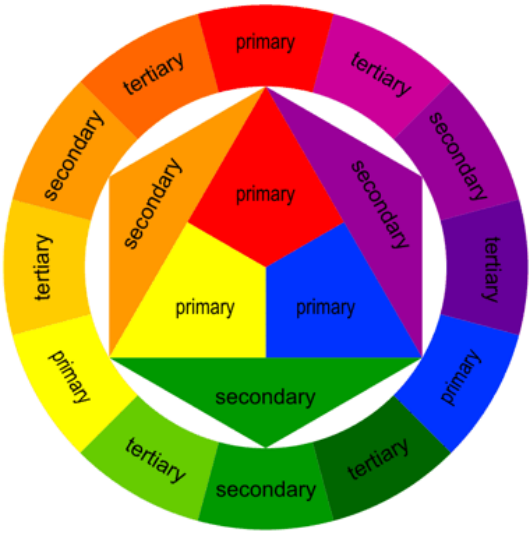


Figure 5. Traditional color theory diagram. The human eye perceives red, green, and blue which are not equally distributed according to color theory. The combinations of color between red and green offer the widest and most distinct set of colors to for viewing the similarities and differences between overlapping images.

The brightness values in various image modalities are essentially arbitrarily and subject to lighting conditions and calibration of the camera used. In order to more accurately compare images, it is beneficial to scale the intensity of each image in order to equalize their brightness. This ensures one image is not overwhelmed by the other and that the full color range from red to green is present in the images. It is also possible to apply histogram based intensity scaling in order to diminish the appearance of strong infrared signals from light pigments, or to increase the contrast in an image in order to make edges more apparent. Adjusting contrast and can improve the clarity of major features in a false color image, but at the expense of diminishing other features. The relative intensity in an infrared image is important when analyzing pigments present in the image. Altering the contrast makes the form of the image more apparent to a viewer, but can obscure other important information. Contrast adjustments should be used with care when analyzing and comparing images.

Figure 6. False color composite images of infrared (green) and visible spectrum (red) images overlaid. (Top Left) The two images are not correctly registered. (Top Right)) False color image in red and blue. It is more difficult to discern features in gradations of purple. (Bottom Left) False color image with unmodified contrast. The exact position of the underlying figure can be seen in relation to the fruit painted over it. (Bottom Right) The same image with increased contrast. The forms in the image are more visible, especially the plant over the seated figure.

Conclusion

Registration by feature point matching is a viable method for registering multispectral images of paintings. Sophisticated feature point detection algorithms such

as SURF make it possible to precisely identify and characterize large numbers of feature points in an image. Sufficient matching and refinement using Nearest-Neighbor algorithms and Random Sample Consensus fitting makes it possible to reliably identify matching key point sets. This process makes automatic selection and rescaling of images possible, and replaces the time consuming task for a human. The flexibility feature point detection and camera angle transformation also makes the image acquisition of more flexible because it is not dependent on a specific imaging system or image acquisition conditions. Digital image registration also has potential uses in a variety of conservation studies which utilize x-ray imaging of physical objects, as well as other disciplines including medical and scientific imaging.

Works Cited

Bay, H. (2006). SURF: Speeded Up Robust Features. *Lecture notes in computer science no. 3951*, 404-417.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Carcagnia, P. (2007). Multispectral imaging of paintings by optical scanning. *Optics and Lasers in Engineering 45*, 360 -367.

Conover, D. M., & et al. (2011). Towards Automatic Registration of Technical Images of Works of Art. *SPIE Computer Vision and Image ANalysis of Art II*.

Fontana, R. (2007). Multi-spectral IR reflectography. *O3A: Optics for Arts, Architecture, and Archaeology*.

Goshtasby, A. (2005). *2-D and 3-D Image Registration .* NJ: John Wiley & Sons Inc.

Liew, S. C. (2001). *Interpreting Otical Remote Sensing Images*. Retrieved 2013, from Center For Remote Iimaging, Sensing & Processing: http://www.crisp.nus.edu.sg/~research/tutorial/opt_int.htm

Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision, 60, 2*, 91-110.

Maitre, H. (2008). *Image Processing.* NJ: John Wiley & Sons, Inc.

In [1]:
```
%pylab inline
```

```
Welcome to pylab, a matplotlib-based Python environment [backend: module://IPython.zmq.pylab.backend_inline].
For more information, type 'help(pylab)'.
```

In [2]:
```
from numpy import *
import cv2
from scipy import optimize
import argparse
import itertools
from pylab import*
from scipy import ndimage as n
import scipy as sp
from scipy import stats
import time
```

In [3]:
```
'''Manipulations of 2-D homographies.

R.G. Erdmann

'''
from enthought.traits.api import *
from enthought.traits.ui.api import *
from enthought.traits.ui.ui_editors.array_view_editor import *

from numpy import (zeros, linalg, dot, mat, array, r_, sin, cos, tan,
                   random, eye, outer, sqrt, arctan2, arctan, r_, c_,
                   ones_like, ones, vstack)

class Homography(HasTraits):
    H = CArray(size=(3,3))

    view = View(Item('H', editor=ArrayViewEditor(show_index=False), style='custom'), resizable=True)

    def __init__(self, H=None):
        self.H = H if H is not None else eye(3)
        d = self.H[2,2]
        if d != 1:
            self.H /= d

    def __str__(self):
        s, ar, theta, delta_theta, x, y, theta_x, theta_y = self.to_geometric_parameters()
        return repr(self.H) + '\n' + str(locals())

    def __call__(self, points):
        '''Transform the given list of 2-d points by applying this homography.'''
        return self.apply(points)

    @classmethod
    def R2(selfclass, thetax, thetay):
        '''Homography to rotate by angles thetax and thetay about the x and y axes.'''
        return selfclass(H=array([[cos(thetax), -sin(thetay), 0],
                                  [sin(thetax),  cos(thetay), 0],
                                  [0          ,  0          , 1]]))

    @classmethod
    def T(selfclass, x, y):
        '''Homography to translate by x, y.'''
        return selfclass(H=array([[1., 0,  x],
                                  [0., 1., y],
                                  [0., 0., 1.]]))

    @classmethod
    def S(selfclass, sx, sy):
        '''Homography to scale by sx, sy along the x- and y-axes.'''
        return selfclass(H=array([[sx, 0., 0.],
                                  [0., sy, 0.],
                                  [0., 0., 1.]]))

    @classmethod
    def P(selfclass, thetax, thetay):
        '''Homography to perspective correct by tilting angles thetax and thetay along the x- and y-directions'''
        return selfclass(H=array([[1.         , 0.         , 0.],
                                  [0.         , 1.         , 0.],
                                  [tan(thetax), tan(thetay), 1.]]))


    @classmethod
    def from_point_pairs(selfclass, from_points, to_points):
        '''Returns a homography for transforming arbitrary points from
           one plane to another.
```

```
        from_points is a list of at least four 2-tuples
        to_points is a list of 2-tuples with the same length as
          from_points.

        If exactly four pairs are given for from_points and to_points, the
        homography returned is exact.  Otherwise, a least-squares estimate
        is returned.'''
        n = len(from_points)
        A = zeros((2*n+1,9), dtype=float)

        for i, (f, t) in enumerate(zip(from_points, to_points)):
            p = r_[f, 1]
            row = 2*i
            A[row,0:3] = p
            A[row+1,3:6] = p
            A[row,6:9] = -t[0]*p
            A[row+1,6:9] = -t[1]*p

        U,s,Vh = linalg.svd(A)
        H = Vh[-1,:].reshape(3,3)
        H /= H[2,2]
        return selfclass(H)

    @classmethod
    def from_geometric_parameters(selfclass, arr):
        '''Convert a set of descriptive geometric parameters to a homography.

        See the documentation of the to_geometric_parameters() method
        for complete descriptions of the meanings of the input
        parameters.
        '''
        s, ar, theta, delta_theta, x, y, theta_x, theta_y = arr
        rx, ry = theta-delta_theta/2., theta+delta_theta/2.
        star = s * ar
        sdar = s / ar
        M1 = array([[sdar*cos(rx), -star*sin(ry), 0],
                    [sdar*sin(rx),  star*cos(ry), 0],
                    [0           ,  0           , 0]])
        M2 = outer([x, y, 1], [tan(theta_x), tan(theta_y), 1])
        return selfclass(M1+M2)

    def to_geometric_parameters(self, return_dict=False):
        '''Convert a homography to an equivalent set of geometric parameters.

           Given an arbitrary homography as represented by a 3x3 matrix,
           determine the following 8 geometric parameters needed to
           reproduce it:

             s, ar              :  scale and aspect ratio
             theta, delta_theta :  mean axis rotation angle and y-x angle difference
             x, y               :  x and y shifts
             theta_x, theta_y   :  x- and y-perspective tilt angles

          The matrix can be reproduced, up to a multiplicative factor, as

          T(x, y) * S(s,s) * R2(theta-delta_theta/2, theta+delta_theta/2) * S(1./ar, ar) * P(theta_x, theta_y)

          with the following definitions:

            def R2(thetax,thetay):
                return mat([[ cos(thetax), -sin(thetay), 0],
                            [sin(thetax), cos(thetay), 0],
                            [0,0, 1]])

            def T(x, y):
                return mat([[1, 0, x],
                            [0, 1, y],
                            [0, 0, 1]])

            def S(sx, sy):
                return mat([[sx, 0, 0],
                            [0, sy, 0],
                            [0, 0, 1]])

            def P(thetax, thetay):
                return mat([[1, 0, 0],
                            [0, 1, 0],
                            [tan(thetax), tan(thetay), 1]])


        '''

        Mnorm = self.H / self.H[2,2]
        x, y = Mnorm[0:2,2]
        tanx, tany = Mnorm[2,0:2]
```

```
        Mr = Mnorm[:2,:2] - outer([x, y], [tanx, tany])
        vx = Mr[0:2,0]
        s_over_ar = linalg.norm(vx)
        vy = Mr[0:2,1]
        s_times_ar = linalg.norm(vy)
        s = sqrt(s_over_ar*s_times_ar)
        ar = sqrt(s_times_ar/s_over_ar)
        Mr[:,0] /= s_over_ar
        Mr[:,1] /= s_times_ar
        cos_th, sin_th = Mr[:,0]
        theta = arctan2(sin_th, cos_th)
        min_sin_ph, cos_ph = Mr[:,1]
        phi = arctan2(-min_sin_ph, cos_ph)
        theta, delta_theta = 0.5*(theta+phi), phi-theta

        theta_x, theta_y = [arctan(dummy) for dummy in (tanx, tany)]
        if return_dict:
            return dict(s=s, ar=ar, theta=theta, delta_theta=delta_theta, x=x, y=y, theta_x=theta_x, theta_y=theta_y)
        else:
            return (s, ar, theta, delta_theta, x, y, theta_x, theta_y)

    def apply_homography(self, H):
        '''Modify this homography by applying a homography to this one, returning self'''
        self.H = dot(H.H, self.H)
        return self

    def apply(self, points):
        '''Apply the homography to an array of points.
        Returns an array of transformed points.
        '''
        result = dot(self.H, vstack((points.T, ones(points.shape[0]))))
        return (result[:-1]/abs(result[-1])).T # RGE ?? absolute values added to remove funkiness, but is it sensible/legal?

    def apply_to_point(self, point):
        '''Apply the homography to an array of points.
        Returns an array of transformed points.
        '''
        result = dot(self.H, array([point[0], point[1], 1.]))
        return (result[:-1]/abs(result[-1])).T # RGE ?? absolute values added to remove funkiness, but is it sensible/legal?


    def symbolic_apply(self):
        '''Apply the homography symbolically.

        Returning a list of sympy symbolic expressions for transformed x and y.
        '''

        from sympy import Matrix, symbols, ratsimp
        X, Y = symbols('x, y')
        m = Matrix(self.H)
        r =  m * Matrix(([X],[Y],[1]))
        tr = map(ratsimp, Matrix(r[:2])/r[-1])
        return tr

    def fast_apply(self, x, y, outx=None, outy=None, verbose=True):
        '''Apply the homography using symbolic simplification and an array kernel.

        This routine first attempts to simplify the homography using
        sympy.  See symbolic_apply.  Then, using pytables "Expr"
        functionality, evaluates each of the symbolic expressions for
        the output x and y.  The inputs can be pytables arrays or
        columns, or can be numpy arrays.

        '''
        from tables import Expr
        sx, sy = map(str, self.symbolic_apply())
        ex, ey = Expr(sx), Expr(sy)
        if outx:
            ex.setOutput(outx)
        if outy:
            ey.setOutput(outy)


        if verbose:
            print 'Transforming x coordinates.'
        rx = ex.eval()

        if verbose:
            print 'Transforming y coordinates.'
        ry = ey.eval()

        return rx, ry
```

```
    def inverse(self):
        '''Returns the inverse of this homography.
        '''
        return Homography(linalg.inv(self.H))


if __name__ == '__main__':
    h = Homography()
#    h.configure_traits()
#    h.print_traits()
    print h.to_geometric_parameters()
    h = Homography.from_geometric_parameters([1, 1, 0.2, 0, 100, 300., 0., 0.])


#     from_points = array([[-1408.        ,  -395.        ],
#                          [ -669.        ,  -487.        ],
#                          [  606.        ,  -554.        ],
#                          [ -543.72938689,  -669.31078224]])

#     to_points = array([[  650.48343904,    33.92917548],
#                        [ 1008.5       ,  -340.5       ],
#                        [ 1984.5       ,  -851.5       ],
#                        [ 1000.5       , -1352.5       ]])


# #     from_points = array([ [0,0],   [1,0],      [1,1],    [0,1] ])
# #     to_points =   array([ [0,768], [1024,768], [1024,0], [0,0] ])
#     h = Homography.from_point_pairs(from_points, to_points)
    print h
#    print to_points
    print '-'*80
#    print h(from_points)

    h2 = Homography(array([[  3.35894045e-01,   1.16895606e+00,   1.40483768e+03],
                           [ -5.15135929e-02,   2.09450037e+00,   7.79320086e+02],
                           [ -4.92313397e-05,   8.77291997e-04,   1.00000000e+00]]))
```

```
(1.0, 1.0, 0.0, -0, 0.0, 0.0, 0.0, 0.0)
array([[  9.80066578e-01,  -1.98669331e-01,   1.00000000e+02],
       [  1.98669331e-01,   9.80066578e-01,   3.00000000e+02],
       [  0.00000000e+00,   0.00000000e+00,   1.00000000e+00]])
{'self': <__main__.Homography object at 0x7f6bc1603d10>, 's': 1.0, 'ar': 1.0, 'delta_theta': 0.0, 'theta':
0.20000000000000001, 'y': 300.0, 'x': 100.0, 'theta_x': 0.0, 'theta_y': 0.0}
--------------------------------------------------------------------------------
array([[  3.35894045e-01,   1.16895606e+00,   1.40483768e+03],
       [ -5.15135929e-02,   2.09450037e+00,   7.79320086e+02],
       [ -4.92313397e-05,   8.77291997e-04,   1.00000000e+00]])
{'self': <__main__.Homography object at 0x7f6bc1603cb0>, 's': 0.75652926036416834, 'ar': 1.8667318742273402, 'delta_theta':
0.07742191613865565, 'theta': 0.006266049541788981, 'y': 779.32008599999995, 'x': 1404.8376800000001, 'theta_x':
-4.9231339660225595e-05, 'theta_y': 0.00087729177193340152}
(0.75652926036416834, 1.8667318742273402, 0.006266049541788981, 0.07742191613865565, 1404.8376800000001, 779.32008599999995,
-4.9231339660225595e-05, 0.00087729177193340152)
[0.980066577841242*x - 0.198669330795061*y + 100.0, 0.198669330795061*x + 0.980066577841242*y + 300.0]
[0.980066577841242*x + 0.198669330795061*y - 157.607457022643, -0.198669330795061*x + 0.980066577841242*y - 274.153040272866]
```

In [6]:
```
import cv2
from numpy import *
from scipy import optimize
import argparse
import itertools
```

In [8]:
```
def getkps(ims, iterations = 0 ,r_threshold = .5 ):
    '''
    Identifies a matching set of keypoints in two images

    returns tuple of  x and y coordinates for matching keypoints.
    '''
    num_features = 10000
    features = []
    FLANN_INDEX_KDTREE = 1  # bug: flann enums are missing
    FLANN_INDEX_LSH    = 6
    flann_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)

    matcher = cv2.FlannBasedMatcher(flann_params, {})
    sd = cv2.FeatureDetector_create('SURF')
    detector = cv2.GridAdaptedFeatureDetector(sd, num_features)

    for im in ims:
        for i in range(iterations):    #halves image size iterations times.
            im = cv2.pyrDown(im)

        kp = detector.detect(im)
        print 'Found', len(kp), 'feature points.'
```

```
        descriptor = cv2.DescriptorExtractor_create('SURF')
        (kp, desc) = descriptor.compute(im, kp)

        #features[filename] = (kp, desc)
        features.append((kp,desc))

    n = len(ims)

    for i in range(n-1):
        kpi, desci = features[i]
        flann = cv2.flann_Index(desci, flann_params)
        for j in range(i+1, n):
            #namej = ims[j]
            #kpj, descj = features[namej]
            kpj, descj = features[j]
            #print namei, namej
            idx2, dist = flann.knnSearch(descj, 2, params = {}) # bug: need to provide empty dict
            mask = dist[:,0] / dist[:,1] < r_threshold

            idx1 = arange(len(descj))
            pairs = array( zip(idx1, idx2[:,0], dist[:,0]) )

            maskedpairs = pairs[mask]

            s = np.sort(maskedpairs[:,1], axis=None)
            redundant = set(s[s[1:] == s[:-1]])
            for b in redundant:
                matches = maskedpairs[maskedpairs[:,1] == b]
                ms = matches[np.argsort(matches[:,2])][1:]     #all but first which is nearest neighbor
                mask[ms[:,0].astype(int)] = False

            maskedpairs2 = pairs[mask]

            p1, p2, ds = maskedpairs2.T
            print str(len(p2)) + ' point matches retained'
            if len(p2) < 20:
                print 'insufficient keypoint matches retained, use lower threshold'

    kp1 = array([kp.pt for kp in array(kpj)[p1.astype(int)]])
    kp2 = array([kp.pt for kp in array(kpi)[p2.astype(int)]])
```

In [9]:
```
def get_homography(kp1,kp2, iterations = 0):
    '''
    Calculates transformation homography to align second input array of input keypoints with the first.

    returns: 3x3 homography matrix which will best transform points in kp2 to align with kp1
    '''
    H, status = cv2.findHomography(kp1, kp2, cv2.RANSAC, 5.)
    I = array(Homography(H).to_geometric_parameters())
    I[4] = (iterations+1)*I[4]
    I[5]= (iterations+1)*I[5]
    J = Homography.from_geometric_parameters(I)
    return J.H
```

In [11]:
```
def resecfromkps(ir,vis, scaledown = 0):
    '''
    Identifies corresponding keypoints in two input images and computes a transformation homography for the second image based

    returns: second input with applied perspective transform
    '''
    kp1,kp2 = getkps([ir,vis],iterations=scaledown)
    hh = get_homography(kp1,kp2,iterations=scaledown)
    geo_param_names = ['s', 'ar', 'theta', 'delta_theta', 'x', 'y', 'theta_x', 'theta_y']
    geo_params = Homography(hh).to_geometric_parameters()
    for i in range(len(geo_param_names)):
        print geo_param_names[i], geo_params[i]
    tran = cv2.warpPerspective(vis,hh, (shape(vis)[1], shape(vis)[0]))    # cv2 uses shape in form x,y, not y,x
    return tran, kp1, kp2
```

In [22]:
```
def conv(ir, vis, iterations = 1):
    '''
    Convolves 2 input images and returns the second image translated for best match

    '''
    irsmall = cv2.pyrDown(ir)
    vissmall = cv2.pyrDown(vis)
    con =irfft2(fft2(irsmall)*conjugate(fft2(vissmall)))
    imshow(con)
    a = argmax(con)
```

```
    a = argmax(con)
    dim = shape(con)
    yshift = -2*(dim[0] - a/dim[1])
    xshift = a%dim[1]
    xshift = xshift
    print 'yshift: ' + str(yshift)
    print 'xshift: ' + str(xshift)
    rolledvis = roll(roll(vis,iterations*yshift,0),iterations*xshift,1)
    compirvis = array(compare(ir,vis)).astype(int)
    compirrolvis = array(compare(ir,rolledvis)).astype(int)
    figure(figsize=(20,20))
    subplot(1,2,1)
    imshow(compirvis, cmap = cm.gray)
    subplot(1,2,2)
    imshow(compirrolvis, cmap = cm.gray)
    return rolledvis
```

In [37]:
```
def false_color(vis,ir, colors = 'rg', eqhist = False):
    '''
    creates a single image with two color bands composed from two input images.

    returns false color image
    '''
    comp = zeros((shape(vis)[0],shape(vis)[1],3)).astype(float)

    if colors == 'bg':
        vc,ic,vtit,itit = 2,1,'Blue','Green'
    elif colors == 'br':
        vc,ic,vtit,itit = 2,0,'Blue','Red'
    elif colors == 'rb':
        vc,ic,vtit,itit = 0,2,'Red','Blue'
    elif colors == 'rg':
        vc,ic,vtit,itit = 0,1,'Red','Green'
    elif colors == 'gr':
        vc,ic,vtit,itit = 1,0,'Green','Red'
    elif colors == 'gb':
        vc,ic,vtit,itit = 1,2,'Green','Blue'
    else:
        vc,ic,vtit,itit = 1,0,'Green','Red'

    if eqhist:
        vis = cv2.equalizeHist(vis)
        ir = cv2.equalizeHist(ir)
    else:
        vis = vis.astype(float)/vis.mean() * 100.0
        irmean = ir.astype(float)/ir.mean() *100.0

    comp[:,:,vc] = vis.astype(int)
    comp[:,:,ic] = ir.astype(int)
    return comp
```

In [12]:
```
ims1 = ['/media/raid/tobin/Dropbox/art research/14PVISIB001_sub.tif','/media/raid/tobin/Dropbox/art research/14IRPHT0001_sub.t
ims2 = ['/media/raid/tobin/Dropbox/art research/example_photos/47MCPVIS_pyr.tif','/media/raid/tobin/Dropbox/art research/examp
```

In [22]:
```
vis = cv2.imread(ims1[0], cv2.IMREAD_GRAYSCALE)
ir = cv2.imread(ims1[1], cv2.IMREAD_GRAYSCALE)
print shape(ir), shape(vis)
```

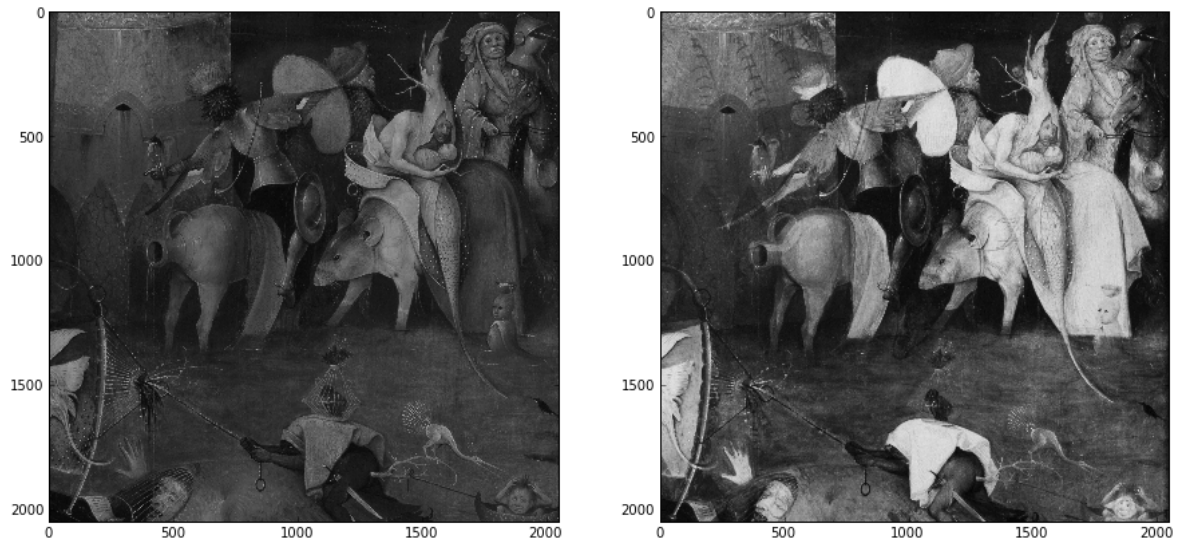
```
(2048, 2048) (2048, 2048)
```

In [ ]:
```
'''









'''
```

In [23]:
```
figure(figsize =(15, 15))
subplot(1,2,1)
imshow(vis, cmap= cm.gray)
```

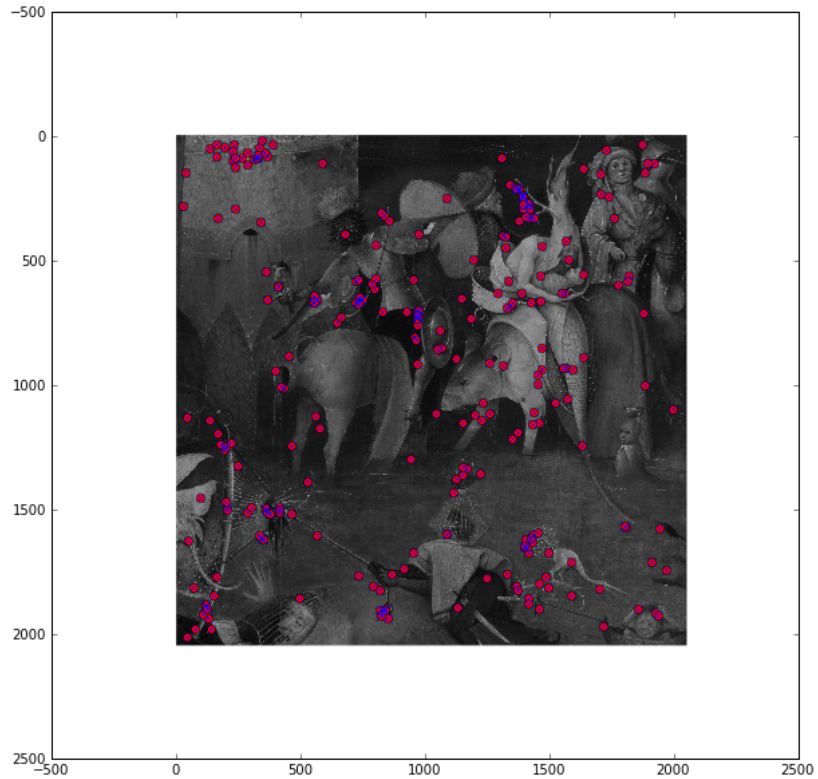
```
subplot(1,2,2)
imshow(ir,cmap= cm.gray)
```

Out[23]: <matplotlib.image.AxesImage at 0x7f5c1c927a90>



In [24]:
```
#Keypoints plotted on visible spectrum image

irkps, viskps = getkps((ir,vis))
shift = [irkps[i]-viskps[i] for i in range(len(irkps))]
figure(figsize=(10,10))
imshow(vis, cmap=cm.gray)
for kp in viskps:
        x,y = kp[0],kp[1]
        plot(x,y,'ro')
for i in range(shape(irkps)[0]):
        x,y = irkps[i] - shift[i]
        plot(x,y,'bx')
```

```
Found 10000 feature points.
Found 9714 feature points.
261 point matches retained
```

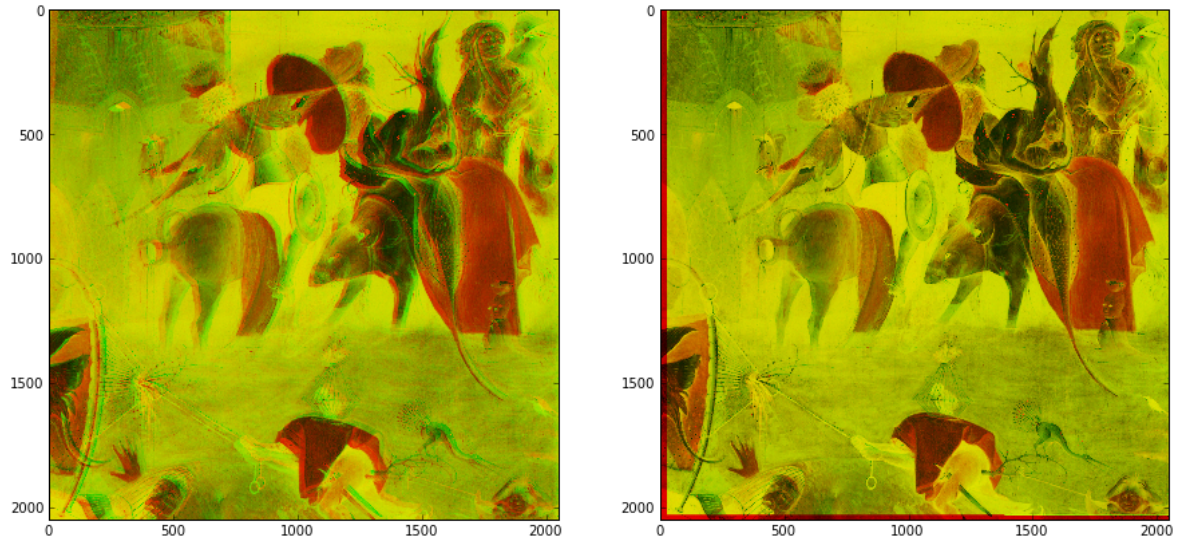


In [41]:
```
#Comparison of images, before and after rescaling
```

```
reir,_,_ = resecfromkps(vis,ir)
figure(figsize =(15, 15))
subplot(1,2,1)
comp = false_color(vis,ir)
imshow(comp)
subplot(1,2,2)
comp = false_color(vis,reir)
```

```
Found 9714 feature points.
Found 10000 feature points.
252 point matches retained
s 0.995543030895
ar 1.00034823171
theta 4.71336389101e-06
delta_theta -0.000832688361718
x 27.4953549352
y -9.23433043765
theta_x 5.71610749861e-08
theta_y 4.17866950395e-07
```

Out[41]: <matplotlib.image.AxesImage at 0x11b17210>



In [45]:
```
names = ['s', 'ar', 'theta', 'delta_theta', 'x', 'y', 'theta_x', 'theta_y']
geoparams = []
hs = array([Homography(i).to_geometric_parameters() for i in u])
for i in range(8):
    geoparams.append(hs[:,i])
figure(figsize=(30,20))
for i in range(8):
    subplot(2,4,i)
    g = geoparams[i]
    title(names[i] + '\nmean:    ' + str(g.mean())+ '\nstd:    ' + str(g.std()))
    pyplot.plot(g, stats.norm.pdf(g,g.mean(),g.std()), 'bo')
```