Final Report 18-551, Spring 2001
Panoramic Image Mosaics
Group 4

Bei Lei, blei@andrew.cmu.edu
Claudio Favi, cfavi@andrew.cmu.edu
Christophe Tournery, ct3@andrew.cmu.edu

May 5th 2001

Contents

1 Problem description

We will address the problem of full view panoramic reconstruction from a video sequence. Panoramas can be used to present a 360 degrees view of a landscape as well as the inside of an apartment or a building.

There already exist many different applications (QuickTime VR, Real VR, Surround video, ...) that can reconstruct a panorama, but they usually take as input a series of fixed pictures, such that the user has to give the information regarding the location of each picture. For example if there are 4 pictures covering the panorama the user has to tell which one is representing the north, south, east and west sides.

We will address the more challenging problem of creating the same panorama, but with the input being a non-structured video sequence. This means that we will not give any hints to our program about the location of each frame of the sequence. Moreover the video camera can rotate horizontally and vertically around its stationary optical center. This implies that the view axis of the camera does not have to stay on the horizontal plane, but the camera should stay at the same point, all along the video sequence. The latter is to avoid to any 3-D structure of the objects to reveal, since our project is designed to support only piece-wise planar scene, as is the case with many building interiors or with outdoor panoramas, but not to recover 3D projective depth. It implies that while shooting the video sequence, if we are indoor, where objects are close to the camera, we will probably need a tripod for the camera, and make sure that the rotation point is right below the CCD (or film). For outdoor videos, where objects are far away from the video camera, the video can be taken "by hand", without a stand, because if objects are far enough, a small translation of the camera will not reveal the 3-D structure of the environment. Of course no moving objects are allowed.

2 Solutions

Many different approaches can be adopted to solve the above mentioned problem. In fact, it comes down to decide:

- how to do images registration,
- how do we do the warping,
- with which strategy we apply the previous 2 points.

Let's explain some terms...

2.1 Terminology

- Mosaicing: is the automatic alignment of multiple image into larger aggregates onto a common reference plane. Image alignment relies on finding corresponding points over a sequence of the scene. This is usually achieved

through an approximation of the 2D motion field, the optical flow, that is the apparent motion of the image brightness pattern.

In this project, direct minimization of discrepancy in pixel intensity is used to align images.

- Transformation: geometric transformation models the motion/distortion between two images: translation, scaling, rotation, shearing, perspective.
- Motion: when we estimate the motion relating two images, we're looking for the geometric transformation.
- Registration: registration is the task of matching two or more images. It's the central task of image mosaicing procedures. In our approach, it consists in computing the transformations relating various scene pieces so we can paste them together. This procedure is applied to gray-scale image for performance reason.
- Background: is the blank big image we allocate at the beginning of panorama reconstruction, it holds the panorama progressively reconstructed from video sequence.
- Warping: given the transformation between two images, warping one image to another(can be a background) means positioning one to the other on a common frame according to the transformation. After this, averaging images based on gray-scale will be needed to composite them.

2.2 Images Registration

We will first focus on three different models for local image registration. Then we will describe how we can use theses models to create a panorama from a video and what are the pros and cons of each model applied to video sequences.

2.2.1 Overview of Registration

The main idea of registration is to find a transformation M that maps every pixel from image I_1 onto image I_0. Because a general model for 3D registration is a 3 by 3 matrix, we use homogeneous coordinates for every pixels of an images. For example, if $\mathbf{x} = \begin{bmatrix} x & y & 1 \end{bmatrix}^T$ is a pixel in image I_1 then its projection by matrix M is:

$$\mathbf{x}' = \mathbf{M}\mathbf{x} = \begin{bmatrix} m_o & m_1 & m_2 \\ m_3 & m_4 & m_5 \\ m_6 & m_7 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

We will now address three different models for the transformation matrix. The first is the perspective (8 parameter) model, the second is a 6 parameter model used in the code from Jose Moura's content-based video processing project, and the last is the rotational (4 parameter) model which gives the best results in term of registration and speed for our algorithm. Even thought the

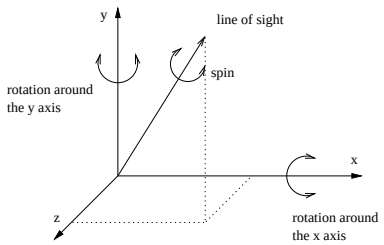


Figure 1: Camera Rotations

4 parameter model gives the best results, we will first focus on the perspective model, because it the most general one and the two others are only restrictive versions of it.

2.2.2 8 Parameters

This model is able to represent all the image transformations a camera motion could create (see Figure 1):

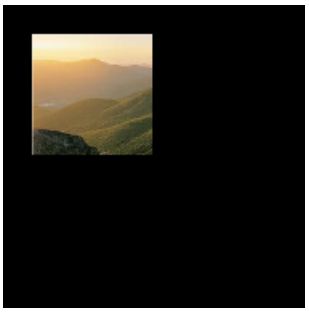
- Zoom.
- Rotation, translation, perspective created by camera rotations around the z and y axis plus camera spin (rotation of the camera around its view axis).
- Translations created by camera translation around the three axis.(10 feet to the north for example), as long as no 3D appears on the image (we assume the translation distance is negligible compared to the distance from the camera to the objects.
- Shearing.

All combination of theses transformations can be achieved using matrix multiplication. Let us illustrate the basic transformations on some examples.
Note: Because the origin of the coordinates for an image is its upper-left corner (meaning that the horizon is at the top of an image), every transformation is applied from this point. This sometimes gives results that we do not expect at the first time, but we can always correct this by changing the origin of the coordinates (matrix multiplication). But to keep the examples simple we stay with the origin at the upper-left corner of the image.
Each transformation matrix shown here is inverted. For example the matrix representing a zoom out by a factor of 2 *should contain "*0.5*" values on the first two diagonal elements and not "*2*". This is not an error but a required feature for re-sampling the images which will be explained in section 2.3 on page 13.*

- Zoom.
  Here is a matrix that does a zoom out by a factor of 2 on an image:

$$\mathbf{M} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  The original image, and the zoomed image:

- Rotations: The general rotations parameters are:

$$\begin{bmatrix} 0 & -\omega_z & f\omega_y \\ \omega_z & 0 & -f\omega_x \\ -\omega_y/f & \omega_x/f & 0 \end{bmatrix}$$

  Where f is the focal of the camera. For the next examples f will be set to 100 to enhance the results on the images. But common values for f are between 500 for a camera and $2000 - 5000$ for a web-cam.

- Rotation around the x axis.
  Here is a matrix representing a rotation of 5 degrees to the left of the camera:

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 8.72665e+0 \\ 0 & 1 & 0 \\ -8.72665e-4 & 0 & 1 \end{bmatrix}$$

  The original image, and the rotated image:


This represent the projection of the image taken by the rotated camera on the plane of the first image (where the camera was not rotated). The content of the images is actually identical because we applied the transformation on the first image, and not on an image taken by a real camera rotated by 5 degrees.

- Rotation around the y axis.
  Here is a matrix representing a rotation of 5 degrees up of the camera:

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 8.72665e+0 \\ 0 & -8.72665e-4 & 1 \end{bmatrix}$$

  The original image, and the rotated image:

  Again this represent the projection of the image taken by the rotated camera on the plane of the first image (where the camera was not rotated). The content of the images is actually identical because we applied the transformation on the first image, and not on an image taken by a real camera rotated by 5 degrees.

- Spin.
  Here is a matrix representing a spin of 10 degrees of the camera:

$$\mathbf{M} = \begin{bmatrix} 1 & -0.17453 & 0 \\ 0.17453 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  The original image, and the rotated image:

The content of the images is actually identical because we applied the transformation on the first image, and not on an image taken by a real camera after a 10 degrees spin. Remember: the origin of the coordinates (and the rotation axis) are at the upper-left corner of the image, not in the middle of the image.

- Camera Translation.
  Here is a matrix representing a pure translation of 18 pixels to the right and 10 up.

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & -18 \\ 0 & 1 & 10 \\ 0 & 0 & 1 \end{bmatrix}$$

  The original image, and the shifted image:

  The content of the images is actually identical because we applied the transformation on the first image, and not on an image taken by a real camera shifted by a few feet.

- Shearing (x and y axis).
  Although this transformation never appears in images taken by a camera, it's very useful to have a model capable of handling the shearing to correct error propagation. See Section **??**.

  Here is a matrix representing a shearing along the y axis.

$$\mathbf{M} = \begin{bmatrix} 1 & 0.1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  The original image, and the sheared image:

This model is very general but suffers two problems to be efficient in our algorithm: it defines camera translations which are not in the purpose of this project, and it introduces more errors when trying to find the good transformation matrix, due to many unknowns to find. We will compare the models more in detail in section 2.2.7.

2.2.3 6 Parameters

The 6 parameters model is a translation model: it defines all the transformations defined in the 8 parameters model, except the perspectives created by camera rotations around the x and y axis. The matrix $\mathbf{M}$ for this model contains 6 parameters:

$$\mathbf{M} = \begin{bmatrix} m_0 & m_1 & m_2 \\ m_3 & m_4 & m_5 \\ 0 & 0 & 1 \end{bmatrix}$$

This is not a adequate model for our problem where we want to reconstruct a 360 degrees panorama, because of the lack of the ω_x and ω_y rotations. But it was the model used in the code from Jose Moura for video compression, and thus it was the first we tried before moving to an eight or four parameters model.

2.2.4 4 Parameters

This model handles the three possible camera rotations (ω_x, ω_y, ω_z) plus a focal f. The matrix $\mathbf{M}$ corresponding to this model contains 8 parameters like the general perspective model, but the parameters are related to each others. The relationship between a 3D point $\mathbf{p} = (X, Y, Z)$ and its image coordinates $\mathbf{x} = (x, y, 1)$ can be described by

$$\mathbf{x} = \mathbf{VRp}$$

where

$$\mathbf{V} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}, \mathbf{R} = [r_{ij}]$$

are the focal length scaling and 3D rotation matrices. Because we have made the assumption that the focal length is the same for all images we are left with

only three parameters to compute instead of six or height. This leads to less errors in the registration phase, or less iterations to achieve the same level of error as with six or height parameters.

2.2.5 Overview of Algorithm - Levenberg-Marquard

The algorithm used to register two images consist in minimizing a least-square problem using the Levenberg-Marquard algorithm. The different steps of the Levenberg-Marquard could be summarized as follow:

1) Initialization.

2) Compute error E_0.

3) Find a better estimate.

4) Compute new error E_1.

5) Update estimate according to errors E_0 and E_1.

6) Goto 2) unless error is small enough or too many iterations.

Now we go in more details about these six steps:

1) The first estimate for the transformation matrix is either the identity matrix for the registration of the first pair of images, or the previous transformation, attenuated by a dumping factor to avoid over-estimation, for the next images.

2) The intensity error is described more in details in section 2.2.6, equation (5).

3) The algorithm used to compute a better estimate is describe in section 2.2.6.

4) We compute the new error E_1 using the new estimate from step 3).

5) Here is the heart of the Levenberg-Marquard method. If the new error E_1 is smaller than the previous error E_0, we keep the new estimate and multiply its diagonal elements by $1 + \lambda$ where lambda is the Levenberg-Marquard parameter. Then we decrease λ for the next step. However if the new estimate gives an error greater than the previous one, we discard the new estimate, multiply the diagonal elements of the previous estimate by $1+\lambda$, increase λ, and use this updated previous estimate as the current estimate.

6) We re-iterate the procedure with the new estimate (either the one computed in step 3) or the previous one multiplied by λ). We stop if the error is below a threshold, or if too many iterations have been made, meaning that the algorithm is unable to register the two images.

In the next section we will develop the maths used to find a better estimate of the transformation matrix for two images (used in step 3)).

2.2.6 Finding a better estimate

Because we end-up using the four-parameters model we will describe the algorithm adapted to it, but the algorithm is very similar for other models.

As we saw in Section 2.2.4, $\mathbf{x} = \mathbf{VRp}$. But the translation matrix $\mathbf{M} = \mathbf{VR}$ still have height parameters, therefore the four-parameters model warps an image into another using: $\mathbf{x}' = \mathbf{Mx}$ where $\mathbf{x} = (x, y, 1)$ and $\mathbf{x}' = (x', y', 1)$ are homogeneous coordinates. This equation can be re-written as

$$x' = \frac{m_o x + m_1 y + m_2}{m_6 x + m_7 y + 1} \quad (1)$$

$$y' = \frac{m_3 x + m_4 y + m_5}{m_6 x + m_7 y + 1} \quad (2)$$

For a camera rotating around its center of projection, the mapping (perspective projection) between two images k and l is therefore given by

$$\mathbf{M} = \mathbf{V}_k \mathbf{R}_k \mathbf{R}_l^{-1} \mathbf{V}_l^{-1}$$

where each image is represented by $\mathbf{V}_k \mathbf{R}_k$, i.e., a focal length and a 3D rotation.

We now assume that the focal length is known and is the same for all images, i.e. $\mathbf{V}_k = \mathbf{V}_l = \mathbf{V}$. To recover the rotation, we perform an incremental update to $\mathbf{R}_k$ based on the angular velocity $\mathbf{\Omega} = (\omega_x, \omega_y, \omega_z)$,

$$\mathbf{M} \leftarrow \mathbf{V}\hat{\mathbf{R}}(\mathbf{\Omega})\mathbf{R}_k \mathbf{R}_l^{-1} \mathbf{V}^{-1}$$

where the incremental rotation matrix $\hat{\mathbf{R}}(\mathbf{\Omega})$ is given by Rodriguez's formula

$$\hat{\mathbf{R}}(\hat{\mathbf{n}}, \theta) = \mathbf{I} + \sin\theta \mathbf{X}(\hat{\mathbf{n}}) + (1 - \cos\theta)\mathbf{X}(\hat{\mathbf{n}})^2$$

with $\theta = \|\mathbf{\Omega}\|$, $\hat{\mathbf{n}} = \mathbf{\Omega}/\theta$, and

$$\mathbf{X}(\mathbf{\Omega}) = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}$$

is the cross product operator. Keeping only terms linear in $\mathbf{\Omega}$, we get

$$\mathbf{M}' \approx \mathbf{V}[\mathbf{I} + \mathbf{X}(\mathbf{\Omega})]\mathbf{R}_k \mathbf{R}_l^{-1} \mathbf{V}^{-1} = (\mathbf{I} + \mathbf{D}_\Omega)\mathbf{M}$$

where

$$\mathbf{D}_\Omega = \mathbf{V}\mathbf{X}(\mathbf{\Omega})\mathbf{V}^{-1} = \begin{bmatrix} 0 & -\omega_z & f\omega_y \\ \omega_z & 0 & -f\omega_x \\ -\omega_y/f & \omega_x/f & 0 \end{bmatrix}$$

is the deformation matrix.

Re-sampling image I_1 with the new transformation $x' \sim (\mathbf{I} + \mathbf{D}_\Omega)\mathbf{Mx}$ is the same as warping the re-sampled image $\tilde{I}_1(\mathbf{x}_i) = I_1(\mathbf{x}_i')$ by $x'' \sim (\mathbf{I} + \mathbf{D})\mathbf{x}$, i.e.

$$x'' = \frac{x - \omega_z y + f\omega_y}{-\omega_y x/f + \omega_x y/f + 1} \quad (3)$$

$$y'' = \frac{\omega_z x + y - f\omega_x}{-\omega_y x/f + \omega_x y/f + 1} \quad (4)$$

To recover the transformation between two images, we estimate the incremental rotation parameters $\mathbf{d}_\omega = (\omega_x, \omega_y, \omega_z)$ by minimizing the intensity error $E(\mathbf{d}_\omega)$ between two images,

$$E(\mathbf{d}\omega) = \sum_i [I_1(\mathbf{x}'_i + \mathbf{d}_\omega) - I_0(\mathbf{x}_i)]^2 \tag{5}$$

where $\mathbf{x}_i = (x_i, y_i)$ and $\mathbf{x}'_i = (x'_i, y'_i)$ (see equations (1) and (2)), are corresponding points in the two images, and $\mathbf{d}_\omega$ is the incremental update parameter which is the same for all pixels. Using equations (3) and (4), $E(\mathbf{d}_\omega)$ becomes

$$E(\mathbf{d}_\omega) = \sum_i [\tilde{I}_1(\mathbf{x}''_i) - I_0(\mathbf{x}_i)]^2 \tag{6}$$

After a first order Taylor series expansion, the above equation becomes

$$E(\mathbf{d}_\omega) \approx \sum_i [\mathbf{g}_i^T \mathbf{J}_i^T \mathbf{d}_\omega + e_i]^2 \tag{7}$$

where $e_i = I_1(\mathbf{x}'_i) - I_0(\mathbf{x}_i)$ is the current intensity or color error, $\mathbf{g}_i^T = \nabla I_1(\mathbf{x}'_i)$ is the image gradient of I_1 at $\mathbf{x}'_i$, and $\mathbf{J}_i = \mathbf{J}_\Omega(\mathbf{x}_i)$, where

$$\mathbf{J}_\Omega(\mathbf{x}) = \frac{\partial \mathbf{x}''}{\partial \mathbf{\Omega}} = \frac{\partial \mathbf{x}''}{\partial \mathbf{d}_\omega} \frac{\partial \mathbf{d}_\omega}{\partial \mathbf{\Omega}} = \begin{bmatrix} -xy/f & f + x^2/f & -y \\ -f - y^2/f & xy/f & x \end{bmatrix}^T$$

is the Jacobian of the re-sampled point coordinate $\mathbf{x}''_i$ with respect to $\mathbf{\Omega}$.

This minimization problem has a simple least-square solution,

$$\left(\sum_i \mathbf{g}_i \mathbf{g}_i^T \right) \mathbf{d}_\omega = - \left(\sum_i e_i \mathbf{g}_i \right). \tag{8}$$

We solve it using *normal equations*

$$\mathbf{A}\mathbf{d} = -\mathbf{b}$$

where

$$\mathbf{A} = \sum_i \mathbf{J}_i \mathbf{g}_i \mathbf{g}_i^T \mathbf{J}_i^T$$

is the *Hessian*, and

$$\mathbf{b} = \sum_i e_i \mathbf{J}_i \mathbf{g}_i$$

is the *accumulated gradient* or *residual*.

2.2.7 Models comparison

- Four parameters:
  - Advantages:

    * Only three parameters to compute at each iteration of the Levenberg- Marquard algorithm (the focal is assumed fixed and known). This requires less computations to set up and solve the equations.
    * Is a general model well adapted to panorama reconstruction, because it models all possible rotations of the camera.
    * Because it models exactly all the rotations of the camera, this model is able to work with low frame rate. On a video sequence where the camera rotates slowly, we were able to use only 1 frame per second, where other models requires 2 or 3 frames per second on the same video.
  - Disadvantages:
    * Although the model require to compute only three parameters, the transformation matrix has eight parameters. This require additional computations when warping an image an the panorama.

- Six parameters:
  - Advantages:
    * The transformation matrix has only six parameters. This allows a slight speed improvement while warping an image on a panorama, because it requires less computation to locate the projection of a pixel.
  - Disadvantages:
    * Is less general and accurate. With the perspective parameters missing, this model, tries to approximate the rotation of the camera by a translation. This give good results if the images are close to each other, such that the approximation of a camera rotation by a translation is correct.

- Eight parameters:
  - Advantages:
    * Is a general model capable of handling all possible rotations of our camera, and more.
  - Disadvantages:
    * Because it is very general, this model introduces more degrees of freedom than necessary. This is actually a bad thing for us. The model often gets stuck in local minima and converges very slowly compared to the other models.
    * Requires to set up and solve eight equations at each step of the Levenberg-Marquard algorithm, and to solve them. This introduce a large number of computation, because each equation is set up using all the pixels of the image.
    * The transformation matrix has eight parameters. This slows down a little bit the warping process of an image.

Conclusion: The four-parameter model possess all advantages of the six and eight-parameters models, except that the transformation matrix has eight. parameters instead of six, but this is not a big problem, and is more than balanced by the fact that it has only three equations to set up and to solve.

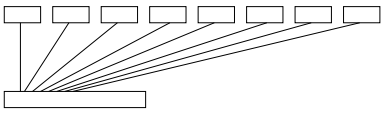
2.3 Warping

Warping is an expensive operation. Warping a 160x120 pixels image to a panorama doesn't just require going over these 160x120 pixels and computing by matrix multiplication with the transformation matrix the corresponding position of this image in the panorama. It actually requires the inverse: go over all the pixels of the panorama and compute, by multiplication with the inverse transform matrix, the position of panorama pixels projected back to the image. Then possible multiple neighbor points on the panorama could take the value of the same image point. The reason for the second approach is to avoid "holes" in the panorama. For example in the case of a zoom (of factor 2), since the transformed image is twice bigger than the original one, if we used the first approach, there will be one blank point every two pixels.

Observing the temporal and spatial locality property of a video sequence, and making assumptions on the camera motion velocity, one could determinate the size of a window in the panorama, over which backward projections are performed, to reduce the number of calculations. This method, not implemented in our program, might offer a speed improvement. However, compare to the execution time each image registration takes ($\approx 7/8$ of time), the warping phase ($\approx 1/8$) is not the bottleneck of speed problem in this project.

2.4 Global Strategy

2.4.1 Sequential Analysis



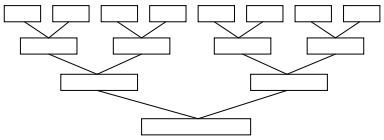
This strategy consists of allocating a big background we estimate large enough at the beginning, then registering sequentially an image with the previous one and warping it to this background immediately after registration.

This approach has the advantage of requiring only one transfer of all the images, the registration with the previous image is faster than with the whole background. But the memory of EVM board will constantly need to hold:

- a background in gray for registration, typically 1500x400x4 bytes.
- a background in RGB for warping, typically 1500x400x4 bytes.
- a copy of the previous image in gray, 160x120x4 bytes
- two copy of the current image in gray and RGB, 160x120x4x2 bytes

Inconvenient: No vertical error correction possible (see section ?? and ?? and no background size adaption possible.

2.4.2 Cascade Analysis

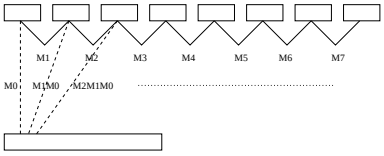


This approach consists of registering pairs of neighbor images in loop, until the number of pairs decreases to zero. This approach doesn't diminish the total error: for example, if with the sequential approach there may be an error in registration between I_0 and $I_1 = e_0$, I_1 and $I_2 = e_1$, I_2 and $I_3 = e_2$, with the cascade approach there might be the same error between I_0 and $I_1 = e_0$, I_2 and $I_3 = e_2$, and e'_1 between the result of the previous registrations. e'_1 is not necessarily less than e_1. Our Unix testing simulation also shows that the second approach doesn't give better quality panorama.

In addition, one step of the loop requires the EVM memory to hold all the warped pairs results, it can consume a significant amount of memory, especially at the beginning of the loop.

This approach doesn't seem to give convincing reason to be used.

2.4.3 Memory Usage Aware Sequential Analysis



This approach consists of registering sequentially the current image with the previous one, but instead of warping it immediately to a pre-allocated background, we just keeps the transformation matrix in a linked list.

At the end of all the registrations, we can compute the necessary size of the background and distribute evenly the error correction on each transformation while reconstructing the panorama by a shearing. The error correction is necessary because, due to the error propagation phenomenon, the panorama often tends to go up or down (see Figure 3 for example).

We chose to use this approach because the two advantages mentioned above, also because of the speed and memory use optimization it offers. Now most of the time the EVM keeps only two copies of gray-scale 160x120 images, only at the final warping phase it keeps a 160x120 RGB current image and RBG background, whose size is adapted from 1500x400 pixels to 1000x100 pixels for the DSP lab sample sequence. This reduction of background size should also improve the warping speed.

The disadvantage of this approach is the re-transfer of RGB images while warping. Since images from the first transfer are destroyed after the registration to spare memory.

2.4.4 Multi-resolution

To avoid getting stuck in local minima and to improve convergence speed while registering two images, we use a technique called multi-resolution. It consists in recursively down-sampling the images and start applying the registration algorithm to the bottom level down-sampled images (see Fig. 2); then using this first approximation of the solution, it refines it by applying the registration algorithm on the upper level and so on. By starting with sub-sampled versions of the images we get an easy to compute estimate which is very valuable for the next step when registering the next level sub-sampled images.

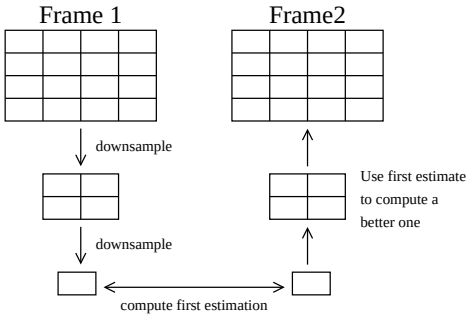


Figure 2: Multi-resolution

2.4.5 Error Correction

It happened that the algorithm would accumulate error, will end with results like in Fig. 3. To cope with such error, we first tried correct it manually applying shearing different shearing on each images of the sequence when warped; we obtained Fig. 4. We also tried to find automatically the correction and this wasn't very successful as one can see in Fig. 5: although the panorama is almost perfectly straight, it's now completely blurred! This is due to the fact that we supposed the correction to be linear and in fact it's not.

3 Implementation

For our implementation we are using the 3 parameters with the "Memory Usage Aware Sequential Analysis" strategy.

Figure 3: Result w/o correction

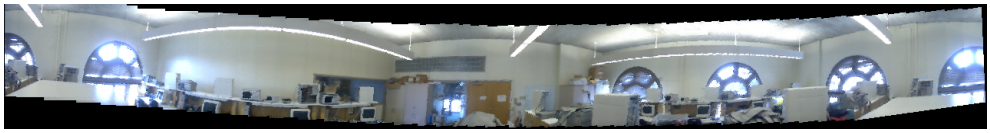


Figure 4: Result w/ manual correction

3.1 Formats

Video Sequence once recorded where converted in a series of still pictures in Portable PixMap (PPM) format (See [1]). We used 160x120 images with color depth of 24bits. For internal manipulation of the images, our program uses a custom C structure define as follow:

```
typedef struct {
     int ncol, nrow;
     IMAGE_FORMAT format;
     PIXEL *data;
} IMAGE;

typedef enum { RGB, GRAY } IMAGE_FORMAT;

/* PIXEL is union type that represent a pixel in different formats */
typedef union {
     struct {
  unsigned char r, g, b;
  unsigned char valid;  /*
 * true if this pixel exists.
 * Used for warping: out of boundary
```

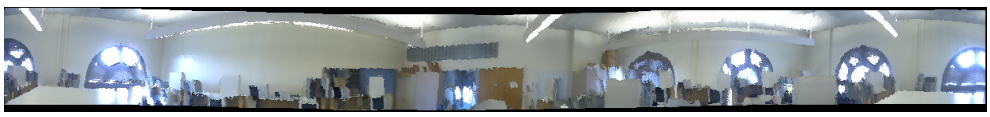


Figure 5: Result w/ auto correction

```
 * pixels are not valid.
 */
    } rgb;
    struct {
  float y;
    } gray;
} PIXEL;
```

3.2 Memory Allocation

The whole text code of our program won't fit on the on-chip memory. In the early stages of development the text would reside entirely on external memory. After some research, we eventually found a way to split the different part of text and locate them in different locations of the EVM memory (See [3] Section 7.8). One can define sections in the command (.cmd) file of the project; those section are then assigned to a certain memory region. Here is part of our command file used for linking our project:

```
/* Allocates different parts into the different areas of memory */
SECTIONS
{
  .vec:          load = 0x00000000 /* Interrupt vector table */

  .clapack:
  {
     dgesv.obj(.text)
     dgemm.obj(.text)
     s_copy.obj(.text)
     s_cmp.obj(.text)
     lsame.obj(.text)
     ilaenv.obj(.text)
     ieeeck.obj(.text)
     idamax.obj(.text)
     dtrsm.obj(.text)
     dswap.obj(.text)
     dscal.obj(.text)
     dlaswp.obj(.text)
     dgetrs.obj(.text)
     dgetrf.obj(.text)
     dgetf2.obj(.text)
     dger.obj(.text)
     xerbla.obj(.text)
  } load > SBSRAM_P

  .rtstext { -lrts6701.lib(.text)
```

```
    } > SBSRAM_P
  .rtsbss { -lrts6701.lib(.bss)
    -lrts6701.lib(.far) } > SBSRAM_P
  .rtsdata { -lrts6701.lib(.const) } > SBSRAM_P

  .libs { -ldrv6x.lib } > SBSRAM_P

  .text:  { *(.text) } load > ONCHIP_P  /* Code */
  .const:   load > ONCHIP_D  /* Variables defined with const */
  .bss:     load > ONCHIP_D  /* Global variables */
  .data:    load > ONCHIP_D
  .cinit:   load > ONCHIP_D
  .pinit:   load > ONCHIP_D
  .stack:   load > ONCHIP_D  /* Stack (for local variables) */
  .far:     load > ONCHIP_D  /* Variables defined with far */
  .sysmem:  load > SDRAM0    /* Heap: malloc and friends */
  .cio:     load > SBSRAM_D  /* printf & co buffer */
  .ipmtext: load > ONCHIP_P  /* Shut the linker up */
  .switch:  load > ONCHIP_P
}
```

3.3 Speed Optimizations

Just by using the memory efficiently, as showed in the previous paragraph, we improved the speed with a factor between 2 and 4. Profiling the program on PC and EVM, we noticed that the programs spent most of its time on operation like applying a function on all the pixels of an image. Here is the result of profiling on Unix (notice that this useful because the execution on the EVM is almost identical than on Unix [besides some rounding errors]):

```
Flat profile:
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 29.04      1.33     1.33      205     6.49    13.20  motion_estimate_affine_step
 27.51      2.59     1.26      220     5.73     5.73  warp_gray
 10.04      3.05     0.46      840     0.55     0.85  filter_5_row
  9.61      3.49     0.44      840     0.52     0.83  filter_5_col
  8.52      3.88     0.39  7140000     0.00     0.00  lpk
  6.33      4.17     0.29        6    48.33    48.33  warp_rgb
  2.62      4.29     0.12  2220000     0.00     0.00  hpk
  2.62      4.41     0.12      220     0.55     0.55  compute_error
  1.53      4.48     0.07        6    11.67    13.33  backgrounder_enter_image
  1.09      4.53     0.05        6     8.33     8.33  image_rgb_to_gray
  0.22      4.54     0.01    77323     0.00     0.00  rgb_same
  0.00      4.58     0.00      226     0.00     0.00  region_clip
```

```
  0.00      4.58     0.00       20     0.00     0.00  image_subsample
  0.00      4.58     0.00       15     0.00   272.43  motion_estimate_lm
  0.00      4.58     0.00        5     0.00   824.00  motion_estimate
  0.00      4.58     0.00        1     0.00   395.00  composePanorama
  0.00      4.58     0.00        1     0.00     0.00  getPanoramaSize
  0.00      4.58     0.00        1     0.00  4175.00  register_images

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
   else blank.

 total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
   function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
   the function in the gprof listing. If the index is
   in parenthesis it shows where it would appear in
   the gprof listing if it were to be printed.
```

NOTE: Some entries were removed from this list because either they were platform dependent (i/o operations) or they were not important for the sake of optimization.

We optimized the functions 'filter_5_row' and 'filter_5_col'. It is important to understand that even if these functions doesn't seem to be the one that take most of time and are not the one called the most, these functions are the one that access all the pixel of the images. Since the bottleneck on the EVM is the memory, by optimizing these function we improve the critical part of the program. Note also the functions 'lpk' and 'hpk'; these are the low/high pass kernel functions and are used in the previously mentioned 'filter_5_*' functions.

For these filter functions , we chose to buffer the input and the output images on the on-chip data memory. We measured the improvement of this optimiza-

tion, in different configurations (text on/off chip). Here are the results:

	w/o Buffering	w/ Buffering
Text Off-chip	24.54	34
Text On-chip	12.45	9.27

In fact the optimization hurt the performance when the text is off-chip. We cannot explain this result without going into the gore of the machine internals, and such discussion is out of the scope of this report. Nevertheless, there are some improvements with the buffering when the text is on-chip.

References

[1] PPM lib
Part of Libgr GNU Licensed graphic library for Unix.
ftp.ctd.comsat.com
Maintainer: Neal Becker (neal@ctd.comsat.com)

[2] http://www.netlib.org/clapack/

[3] TMS320C6000 Assembly Language Tools User's Guide

[4] R.Szeliski: Major reference in image mosaicing and environment. Creating Full View Panoramic Image Mosaics and Environment Maps : General discussion of background reconstruction. http://www.acm.org/pubs/articles/proceedings/graph/258734/p251-szeliski/p251-szeliski.pdf

[5] The Levenberg-Marquard Method
http://www-fp.mcs.anl.gov/otc/Guide/OptWeb/continuous/unconstrained/-nonlinearls/section2_1_2.html

[6] 360-degree photo system by Be Here Corporation http://www.behere.com

[7] iMove-SVS 2000 spherical video by iMove http://www.smoothmove.com

[8] QuickTime VR by Apple http://www.apple.com/quicktime/qtvr

[9] pixAround by Video Ware www.pixaround.com