# Final Project – Image Mosaicing

Abhishek Tripathi
U0562967

## Overview:

The project aims at developing a tool to create a mosaic from a series of images.
The project incorporates various techniques to most accurately align the images with each other.
Before this project, I had been working only on grayscale images for the previous projects. This project has been developed to handle color images as well.

## Mosaicing -Manual Landmark Selection

In this method the user selects the landmarks in the pairs of images using the mouse.

### Implementation details:

Step 1: As implemented in Project 3, the affine transform is calculated by setting up a linear system and solving it to find the affine transform between the pairs of images.

Step 2: Once these transforms have been found, set the middle image I the sequence as the reference image. The transform for this image will be identity.

Step 3: Now set the appropriate transformation matrices for each image, relative to the reference image. Used Cascading of the transforms by multiplying the affine transforms and obtaining the relative transform.

Step 4: Calculate the size of the transformed images by transforming the four corners of the original image. This helps in determining the size of the canvas so that the mosaiced images can be displayed correctly.

For each image:
Step 5: Apply contrast stretching so that there are no sudden changes in intensities of any pair.

Step 6: Apply the transforms and bilinear interpolation to obtain the transformed image intensities. If there is an overlap, blend the overlapping images together using linear blend.

### Algorithm Description:

Landmark Selection:

- Implemented a module that determines a pixel position with the mouse.
  Used the Matlab example (http://www.mathworks.com/help/techdoc/creating_plots/f10-21736.html#f10-11017) to create the module **mousepick (I)**

  The module works as follows:
  To pick a landmark use LEFT CLICK.
  To select the last point use RIGHT CLICK
  The module returns an array of points picked by the user.

- **Use this module to create sets of corresponding pixel positions in a source and a target image.**
- **Set up the linear equation system and implement a solution to solve for the affine transformation between the source and target image.**

The system of equations is calculated using the following steps:

Step 1: Calculate the center of mass for the landmarks from source and target image – Xmean & Ymean

Step2: Find the matrices: $X\sim = X_i - Xmean$ & $Y\sim = Y_i - Ymean$

Where, Xi and Yi are the set of landmarks from source and target images respectively.

Step 3: Find the matrix A by using the equation below:

$$A = \sum( Y_i\sim * X^t_i\sim) *(\sum( X_i\sim * X^t_i\sim))^{-1}$$

Step 4: Find the translation t by using the following equation:

$$t = Ymean - A*Xmean$$

Step 5: Calculate the affine transform using the results found above.

Some techniques applied to improve the quality of results:

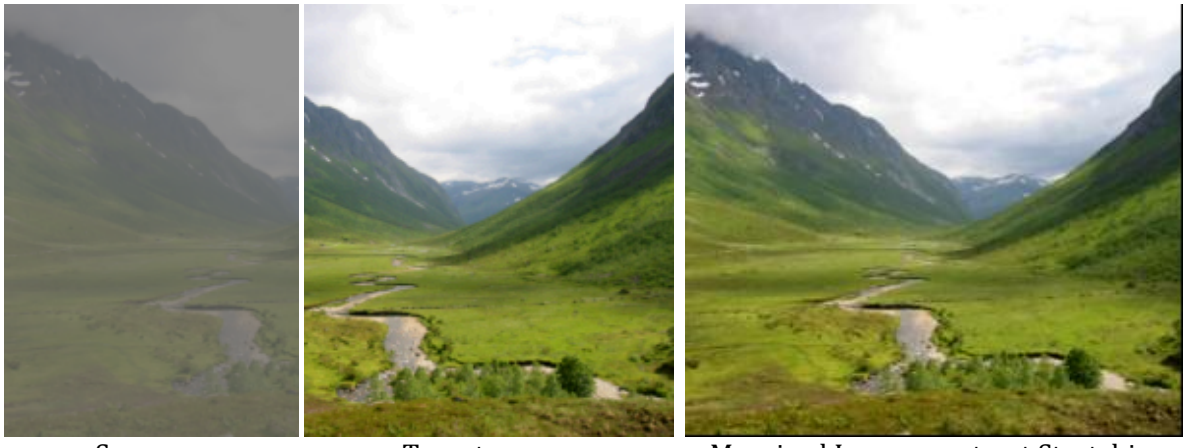
**Contrast Stretching:**

Contrast stretching was implemented and applied on images before they are mosaiced together so that there are no sudden changes in the intensity of the stitched images. I use the following algorithm to implement contrast stretching:

Step 1: Find the minimum and maximum intensity in the gray scale image.
Step 2: get the R, G, and B values for the each pixel.
Step 3: Normalize the R, G, and B -intensity values to the range [minimum, maximum]

**Results:**

In the result below the source image has a poor contrast while the target image has a correct contrast. When contrast stretching is applied the resulting images does not show the jump in contrasts. It enhances the output and corrects the contrast problem as well.



Source Target Mosaiced Image -contrast Stretching

**Feathering:**

I implemented a very basic method to remove the sharp edges. I applied a linear blend of the images at the edges so the sharp edges are smoothed out.

**Image Size:**

Implemented a scheme that takes into account the size of the transformed images and then calculate the size of the final mosaic canvas. This gives us a canvas big enough to accommodate all the images.

This is done as follows:
Step 1: for each image transform the corners to get the end points of the new image.
Step 2: Now find the minimum and maximum in both x & y directions amongst the corners of the transformed images. This gives us the corners of the canvas.

**Cascading:**

Implemented a scheme as follows:

Step 1: Set the reference image as the middle image. The transform of this image is set to Identity matrix.
Step 2: Find the transform of the images paired with the reference image.
Step 3: Find the relative transform of the images paired with the neighbors and multiply them with the transform found in step2.

Traverse until all images have been visited.
This ensures that each transform is relative to the reference image and not just the original images.

**Bilinear Interpolation:**

The bilinear interpolation is done by the following method
Step 1: Calculate the transformed point in the target image using the affine transformation as mentioned in the above section.
Step 2: Find the 4 nearest grid points, eg is the calculated value is (2.3, 4.4) the nearest grid points are P1 (2,4), P2 (3,4), P3 (2,5) & P4 (3,5)
Step 3: Calculate the weights (linear distances in x and y direction)
Step 4: Use the equation below to calculate the value at the pixel:

$$[\{ I(P1)* wx + I(P2)*(1-x) \}*y ] + [\{ I(P3)* wx + I(P4)*(1-x) \}*( 1-y ) ]$$

Where, I(Pn) = Intensity at location Pn
x = weight in x direction
y = weight in y direction

**Questions:**

- **How many control points does it take to get a "good" transformation between images?**

  This is a very subjective question. The "good" transform is a relative term. Hence, I decided to judge the "goodness" of the transformation on the following terms.
  - The transformed image should match the landmarks on the source image.
  - The transformation should align the remaining image in a reasonable threshold. (i.e. the image should not get warped in an unexpected manner.)

Here are some results of the experimentation I did.

4 control points:

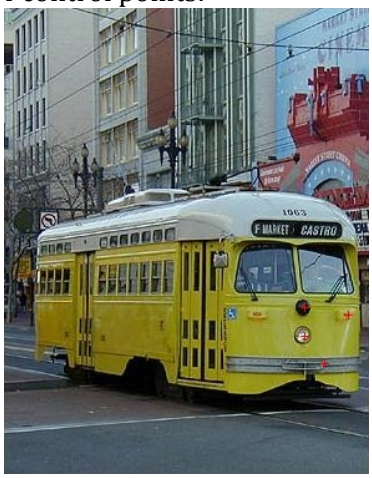



Image1

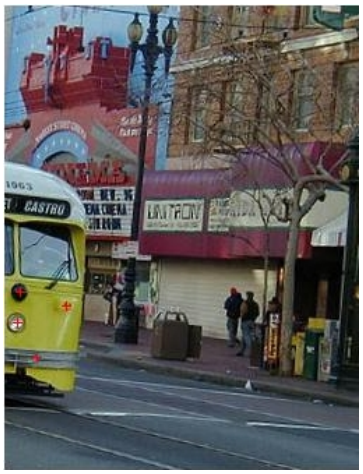
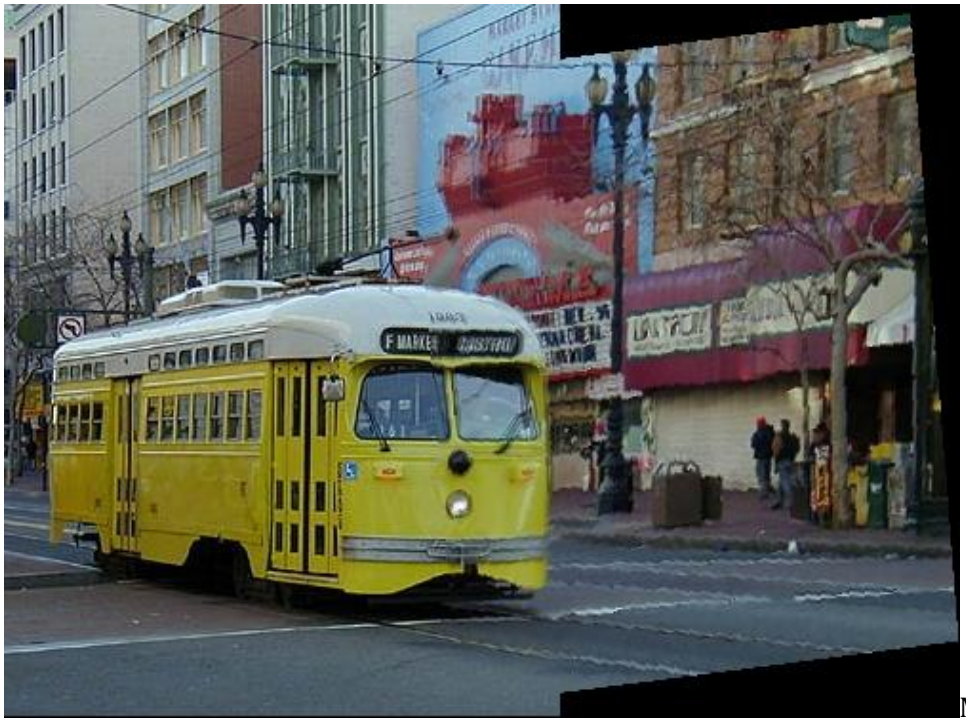



Image2




Mosaiced Image

**Observations:**
The control points match pretty satisfactorily
The image is warped to a nearly accurate transform.
The points other than the control points do not align to a good extent. Specially, the upper part of the tram.

Having 4 control points did not prove to be enough for a “good” transform.

**6 Control Points:**

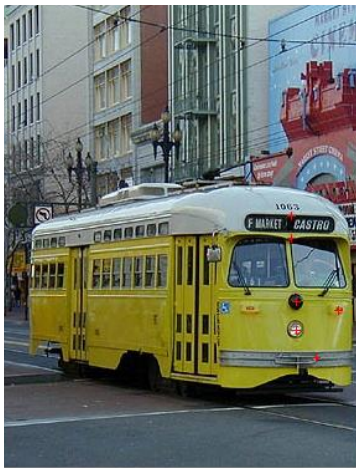



Image1

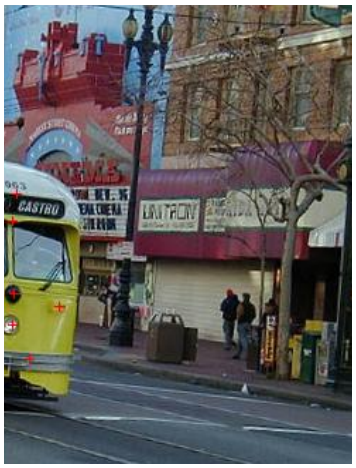
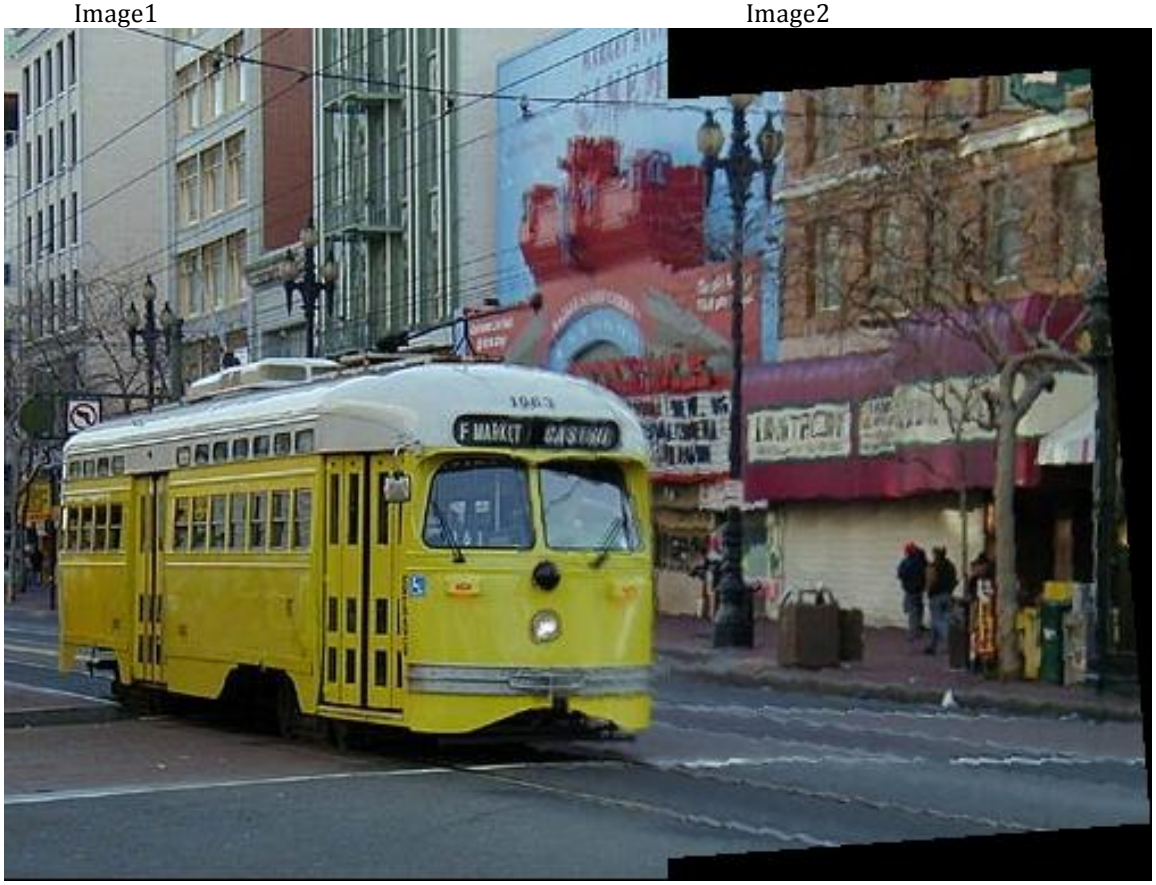



Image2




Mosaiced Image.

Observations:
The control points match pretty satisfactorily
The image is warped to a nearly accurate transform.
This image is much better aligned than that with 4 control points. As we can see the upper art of the tram is overlapping in a better fashion. This shows that 6 control points certainly gives a better transform than 4 control points.

10 control Points.

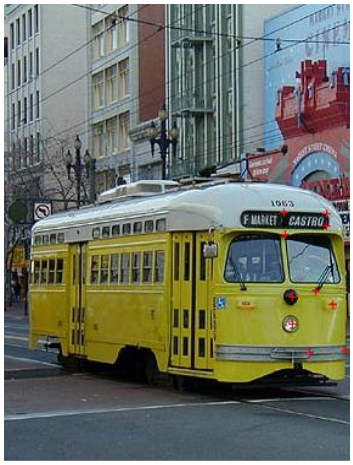



Image1

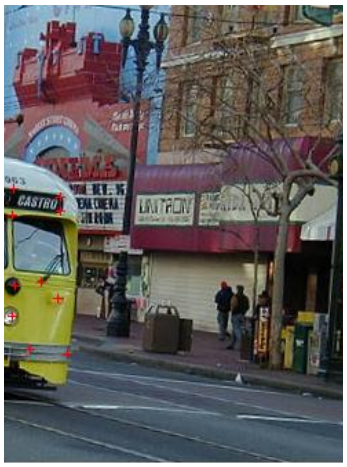
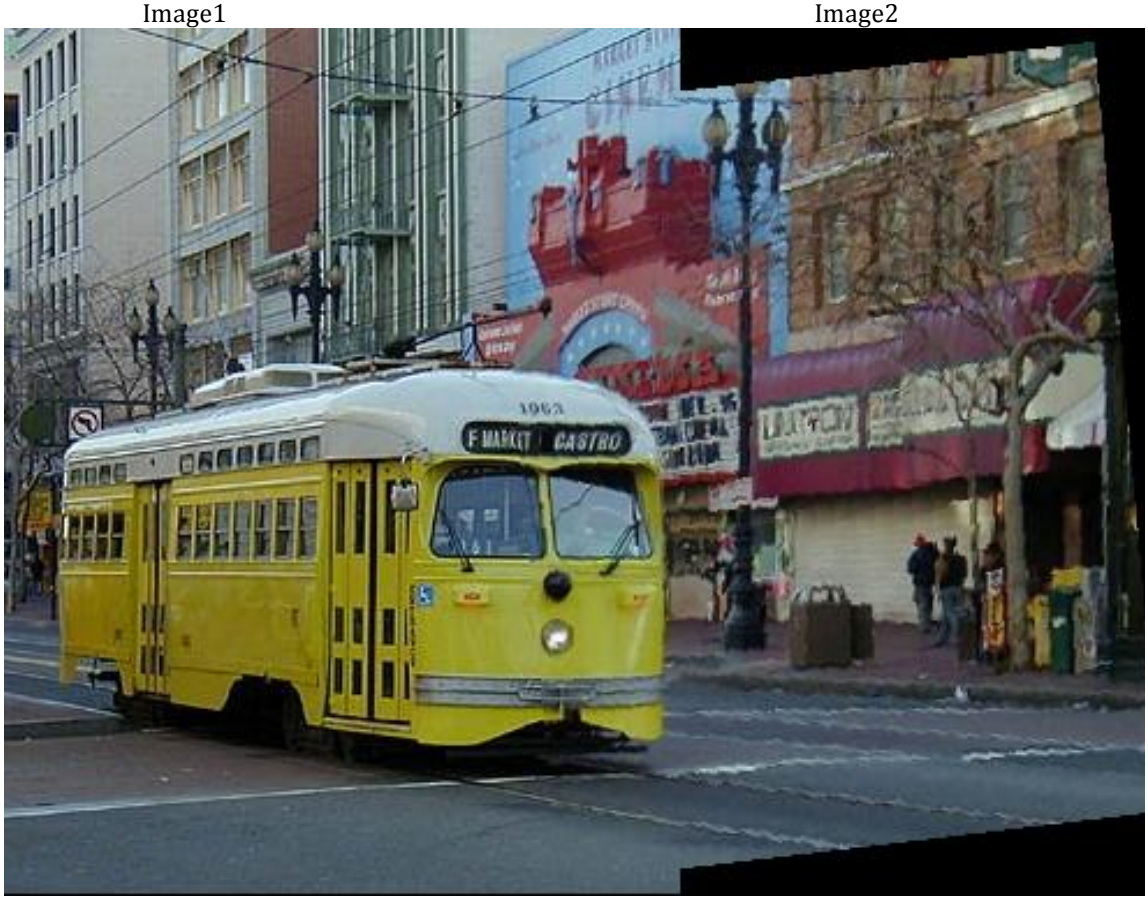



Image2




Mosaiced Image

Observations:

The control points match perfectly.
The image is warped to a “good” transform.
The image is almost perfectly aligned. The signboard of the tram is easily readable in this image, while in the previous images it was not getting aligned properly.

**Choosing more control points definitely improves the quality of the transform.**
10 control points on this image provide a “good” transform.

Note that there is some distortion as the landmarks are handpicked and may not be absolutely accurate. If we do neglect this error then the results are very satisfactory.

Again, the number of control points required for a good transform hugely depends on the following factors:
1.The type of image.
2. The features in the image. If there are some distinct features in the images to be mosaiced then it is much easier to manually select the landmarks as well as the user can be more accurate about them as well.
3. The amount of overlap in the pairs of images is also an important factor. If the overlap is bigger then we have more freedom of selecting the landmarks.
4. Another important factor is the choice of landmarks. The placement of the landmarks plays an important role in the final outcome.

- **How does the algorithm behave at the theoretical minimum of the number of control points?**

The algorithm behaves reasonably well at the theoretical minimum - 4 - of the number of control points. The transform is largely dependent on the accuracy of the control points selected.

With just 4 control points averaging out the errors that creep in as a result of in-accuracy are not eliminated. Hence there might me some distortions.
Globally the transform works well, some local features might not get matched properly due to this.

Below are some examples:
Croydon Panorama – (photo from [www.Eyerex.co.uk](http://www.Eyerex.co.uk))

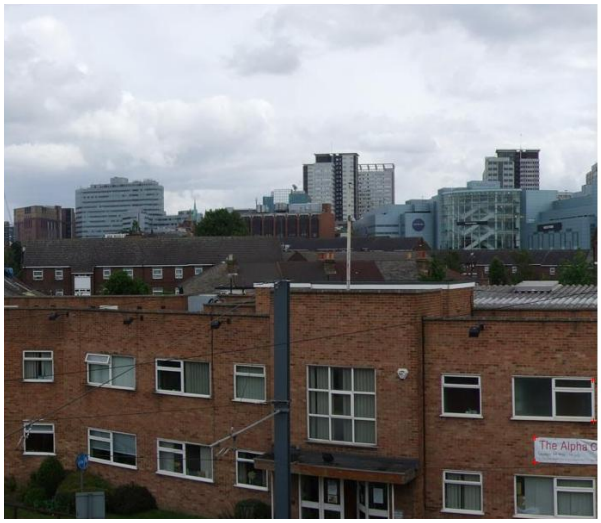
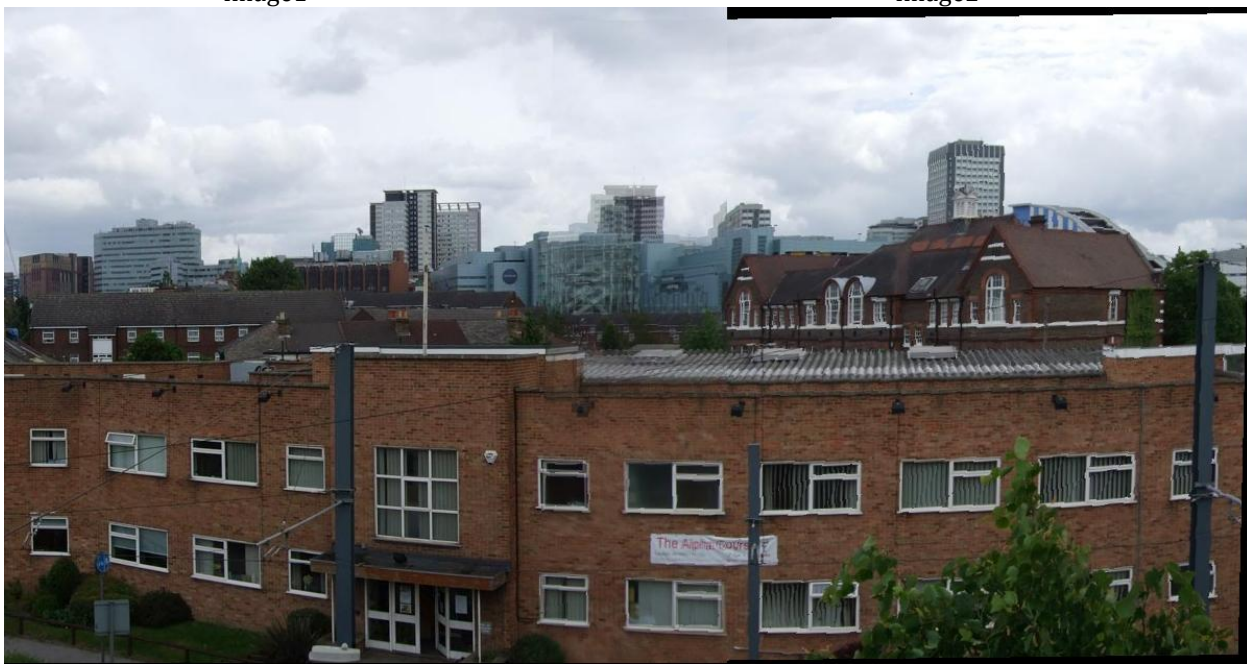


Image1

Image2



Mosaiced Image

Here the banner on the building and the window's match pretty well. I feel the distortions are due to the errors in selecting the landmarks by hand.

New Orleans Lake

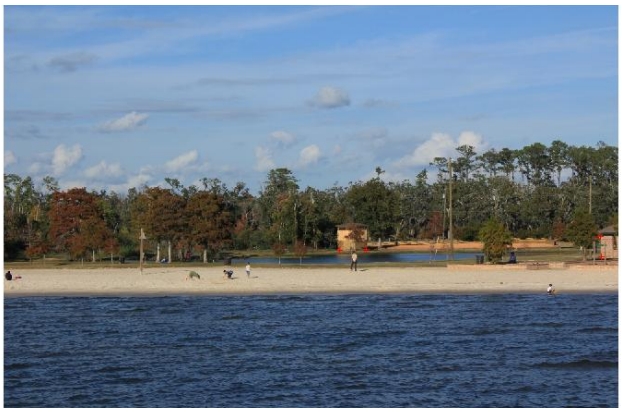


Image1

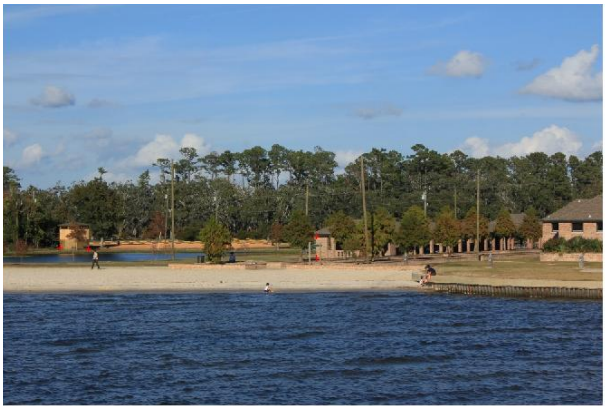


Image2

Mosaiced Image.

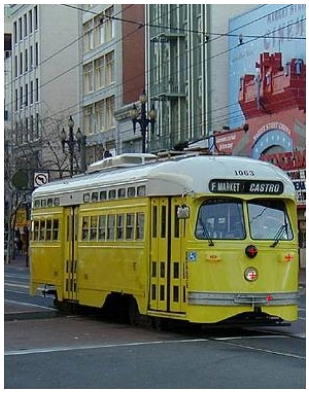
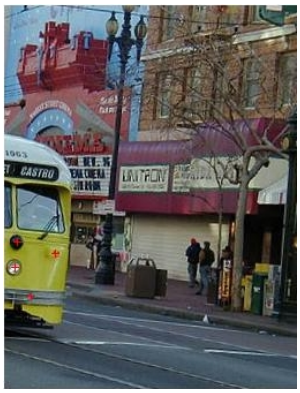
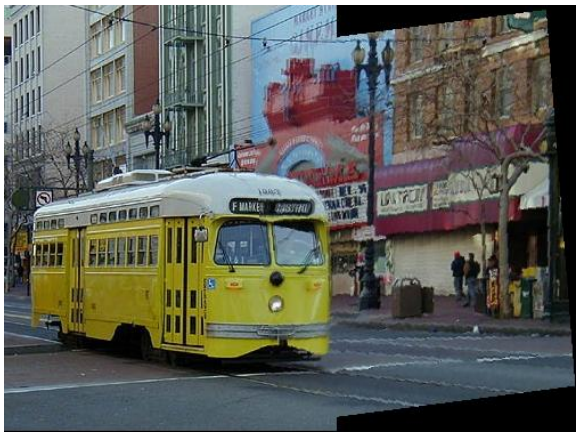
In this image also the distortions are due to the error introduced due to manual selection of the landmarks. The global transform is pretty accurate.

- **From your experiments how does the accuracy of the control points affect the results?**

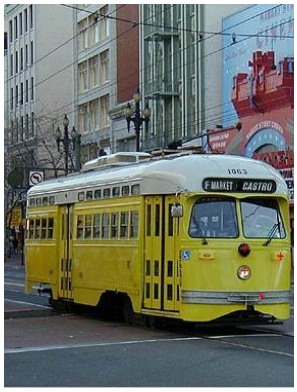
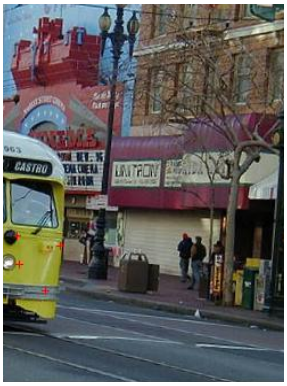
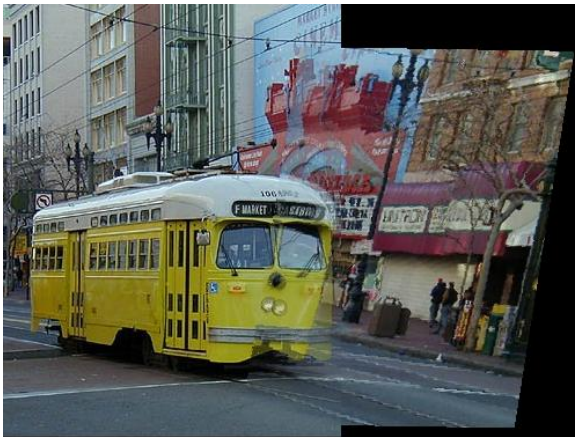
The accuracy of the control points is the most important factor in determining the transform. The accuracy determines how well your transformed image is aligned with the original image.

After rigorous experimenting, I have come to a conclusion that even with the theoretical minimum number of control points we can achieve a good warping if the control points are selected accurately.

In order to show this behavior I have exaggerated the in-accuracy.

Source Target Accurate transform

Source Target In- Accurate transform

As we can see in the above examples that the accuracy of the control points can vastly change the output. It is therefore very important to keep in mind the following, in order to get a "good" transform.

- Choose the Control Points carefully and accurately.
- Choose Control Points sufficient enough to define the most important features in the images to be mosaiced together.

## Extra task:

## Towards Automatic Mosaicing:

I used the SIFT feature extraction method to get the landmark points automatically.
I used the VLFeat 0.9.9 for this purpose.
VLFeat gives an array of control points, which are possible candidates for landmarks.
In order to get a best estimation of the landmarks, it is essential to remove the outliers, which might distort the transform. In order to accomplish this task, I implemented a method **removeOutliers()**
This works as follows:

Step 1: Randomly select "N" control points from the list of control points.
Step 2: Find the transform using the points found in step 1.
Step 3: Using this transform find the target points.
Step 4: Find the difference between the distance of the target points and the points given by initial list.
Step 5: This difference should be minimum in order to find the best transform.
The above steps are repeated for number of times to obtain the best transform.

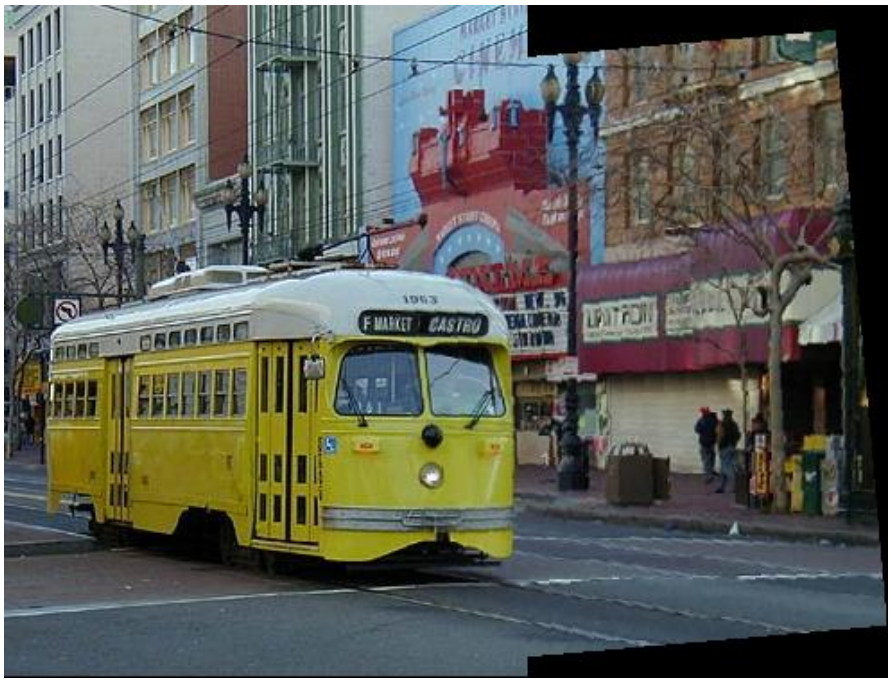
Once the transforms are found for each pair of images. The processes mentioned above are applied to get the mosaiced images.
Here are some of the results of the panoramas formed using automatic mosaicing.
Craydon:
4 control points:


10 Control Points:

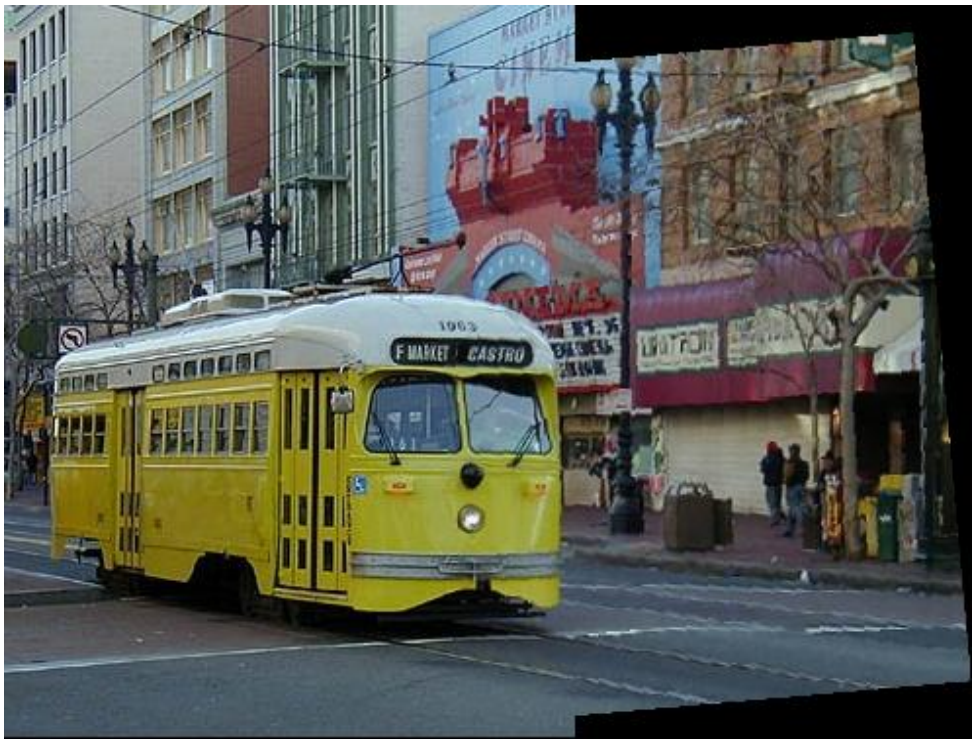
New Orleans Lake:
4 Control Points:

10 Control Points:

Tram:
4 Control Points:



10 Control Points:




It is evident from all the above images that the theoretical minimum number of control points is sufficient provided the control points are accurate. Using the SIFT landmark extraction technique provides with numerically accurate control points, while the process to remove the outliers ensures that the error is minimized. As a result we obtain an accurate set of points to get the transform.

**Limitations of the project:**

If time permits I would like to add the following features:

The project is currently designed for a series of images to create a panorama. I do automatic pairing of the images. It would be more appropriate if I add user defined pairing as a feature. This would allow for mosaicing images with horizontal overlap as well. My unfamiliarity with Matlab's file handling features and lack of time were the factors behind this feature not being implemented.

I would also like to improve the feathering algorithm used in the project.

Also, I would like to speed up the project execution times.

**Code Distribution:**

To run the code use **MyMosaic (auto)**
**auto = 0 for manual mode**
**In manual mode –**

1. **Enter the number of images.**
2. **Enter the path of one image at a time.**
3. **The images appear in pairs. Select the landmarks on the images. Right click to select the last landmark.**

**Result is displayed and stored as "myMosaic.jpg"**

**auto = 1 – Automatic Mode.**

**In Automatic Mode:**

1. **Enter the number of images.**
2. **Enter the path of one image at a time.**
3. **Enter the number of control points you want to be considered for mosaicing.**

**Result is displayed and stored as "myMosaic.jpg"**

**Functions:**
**getTransform (I1, I2)** – Get transform between images I1 & I2. Used in Manual Mode.

**calcTransform (X, Y)** – calculates transform based on the source points (X) and the corresponding target points (Y)

**contrastStretch (Image)** – used to get the contrast stretched image.

**removeOutliers(xa,xb,ya,yb,n,I1,I2)** – removes the outliers from the automatically extracted control points.

**Images:**
**Tram:**
left.jpg , right .jpg
**Craydon :**
c1.jpg , c2.jpg , c3.jpg , c4.jpg
**New Orleans Lake :**
1.jpg, 2.jpg, 3.jpg ,4.jpg.
**Contrast Stretch:**
src1.png , tgt1.png