Project-Report:

Image Mosaicing

**INTRODUCTION:**

Many a time, it is not possible to capture the complete image of a large scene in a single exposure as most of the image capturing media work with images of definite size because of their inherent limit. Here we propose a technique for capturing the complete view of auditoriums and classrooms using less number of cameras .

**APPROACH USED:**

Mosaicing is blending together of several arbitrarily shaped images to form one large radiometrically balanced image so that the boundaries between the original images are not seen. Any number of images can be blended together along user-specified cut lines. This report is basically about image stitching. In this work, we formulate stitching as a multi-image matching problem, and use manually controlled feature points to find matches between all the images.

**WORK PERFORMED:**

***Requirements:***

* Files

This project requires files for matching points explicitly named “new.txt” and

“old.txt” for matching points. Input in this file is required to be in <x y> format.

* Images

This project requires input images named “m0.jpg”... “m1.jpg” and upto 6

images is hard-coded in the code.

***Implementation: Functions used-***

* getPoints:

This function takes a file as an input and return vector (array) of all the points

in <Point2d> format.

* computeHomography:

This function computes Homography matrix using two set of points using svd.

* getCoordinates:

This function computes points <x’,y’> on the transformed image from points

<x,y>

* getLimits:

This function computes the range of the image for offset matching.

* projectImage:

This function computes the warped image using inverse transformation and

finally translates it to match with the final panorama image.

* stitch:

This function fills the panorama with different image and blend them together.

***Implementation: Algorithm***

In our code, we first compute the Homography matrix and based on this matrix we first find the corner limits for all images. Based on these limits, we conclude size of our final mosaic image.

Using Inverse transform, we transform our image to same plane of reference image, and then translate it using offset calculated in above step. Since it’s the minimum and maximum bound for all the images, no image can go outside this range, and we finally obtain warped images.

Finally, we blend these images one by one to compute mosaic image.

***Techniques Used:***

* Feature point extraction

Feature matching matches common points in two images that helps in computing homography matrix. Algorithms like SURF/SIFT can be used to detect and match features in images. Here we’ve done manual matching between two images.

* Image registration.

It is the process of aligning two or more images of the same scene. This process involves designating one image as the reference image, also called the fixed image, and applying geometric transformations or local displacements to the other images so that they align with the reference. Commonly, images are captured under variable conditions that can change the camera perspective or the content of the scene.

* Homography computation

Given enough points (at least 4), we can find the homography matrix using SVD (Singular Value Decomposition).

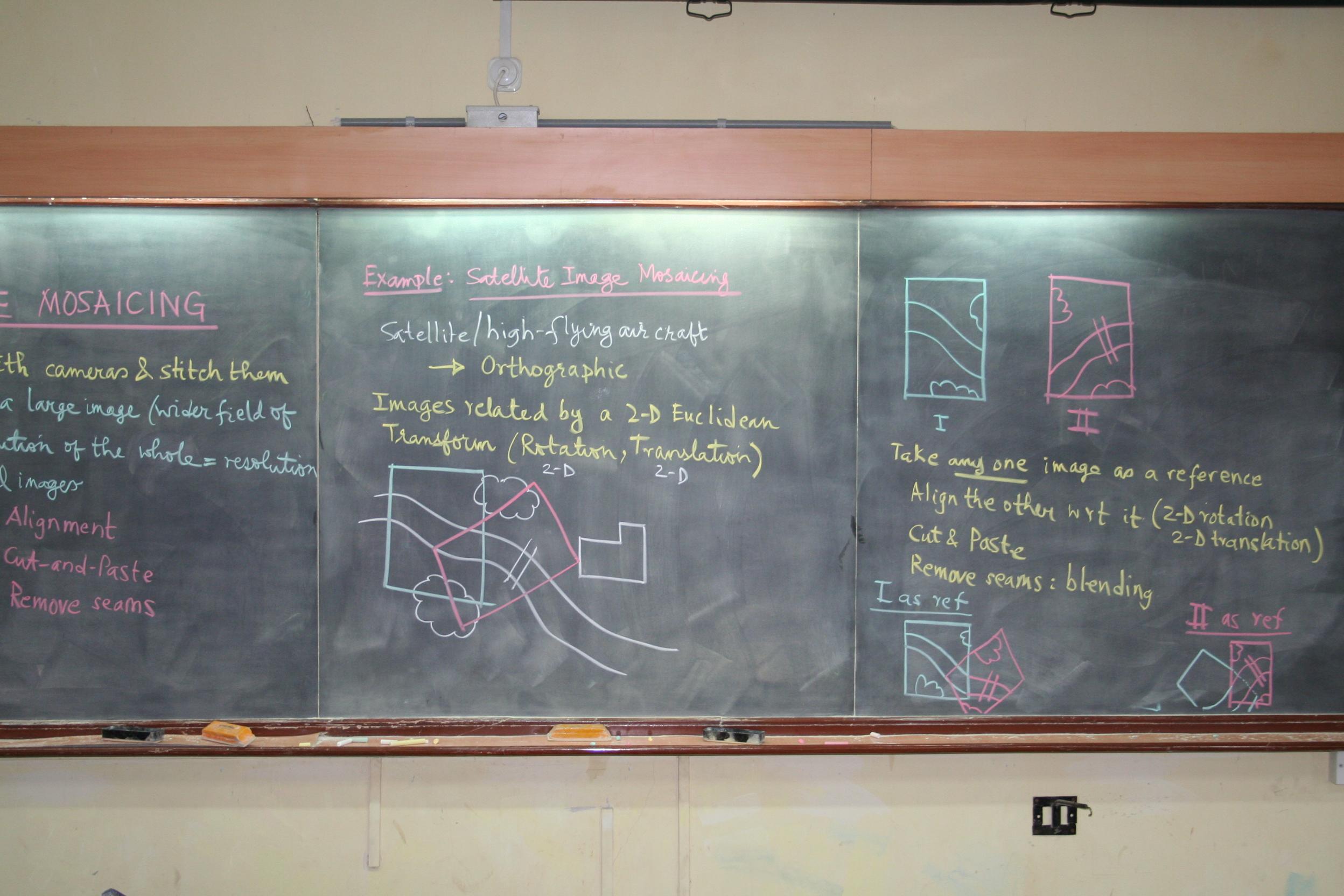
* Warping

We used the previously computed homography matrix and through corner registration, we find the new size of the warped image. Using image homography matrix, we find each point (x’,y’) in the image(2) that locates in image (1) (x,y). On finding the (x,y) we assign the [r g b] color values of image (1) to the pixel located at (x’,y’) in image(2) with some transaction for image bounds.

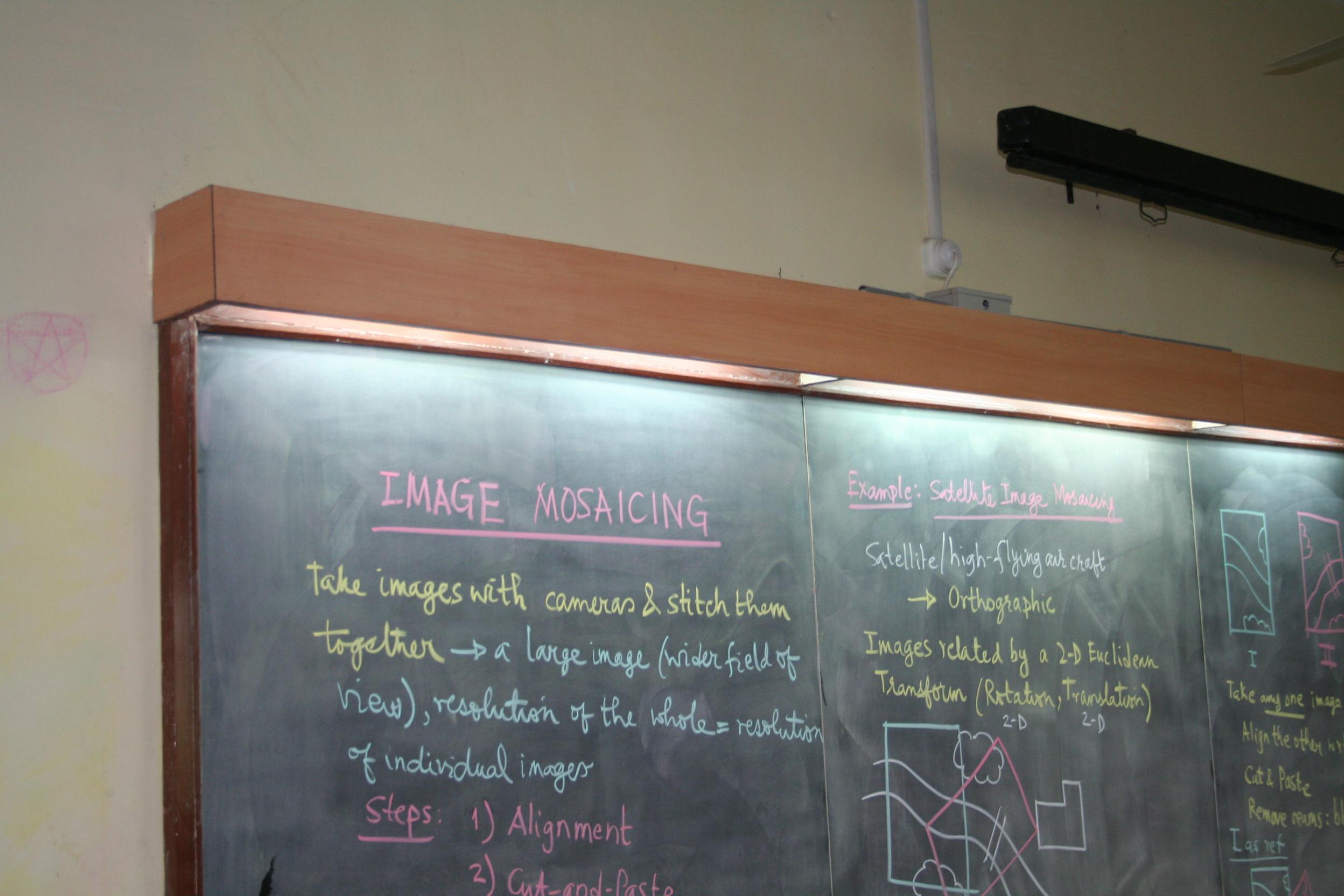
* Image Blending

We did further further translation in the images to map the right point in both images distorted due to translation during wrapping in maintain image bounds. Blending the image result in mosaic of both the images(here just putting one image over the other). As there is only two images as input our blended images and final mosaic image is same. We can further do Histogram Equalisation for better result over change in brightness.

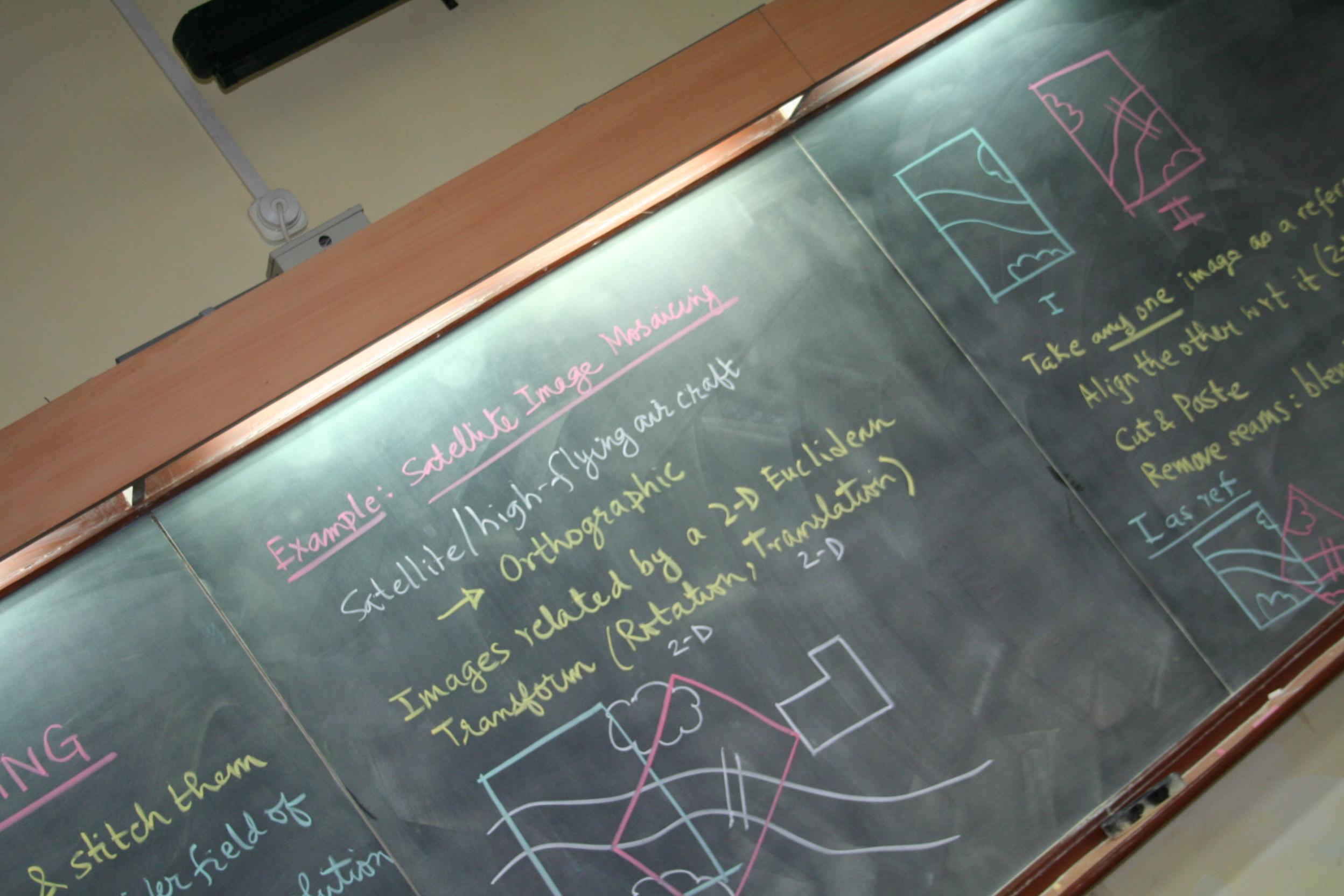
**INPUT IMAGE**

****

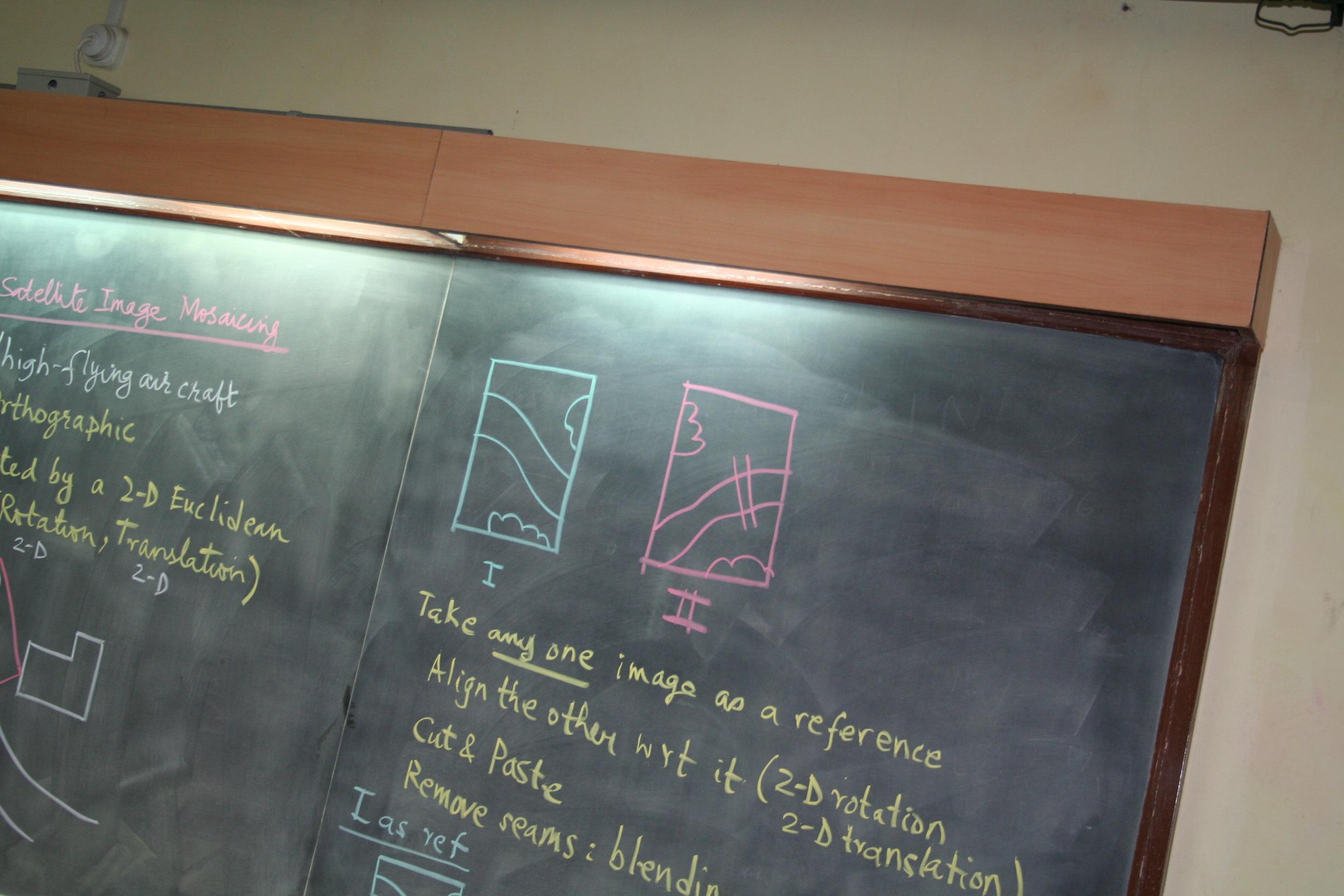
*Mref.jpg*

**

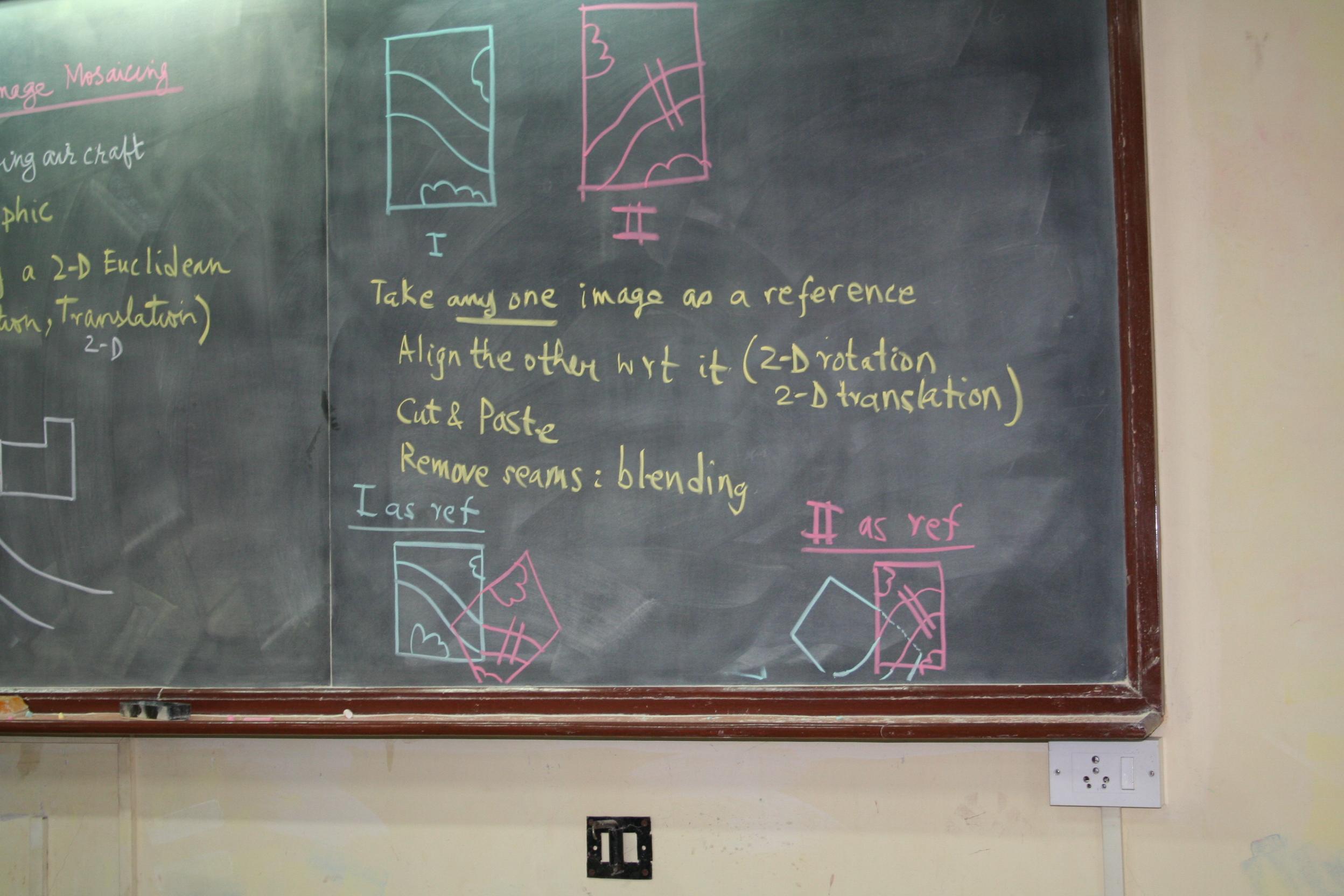
*M1.jpg*

**

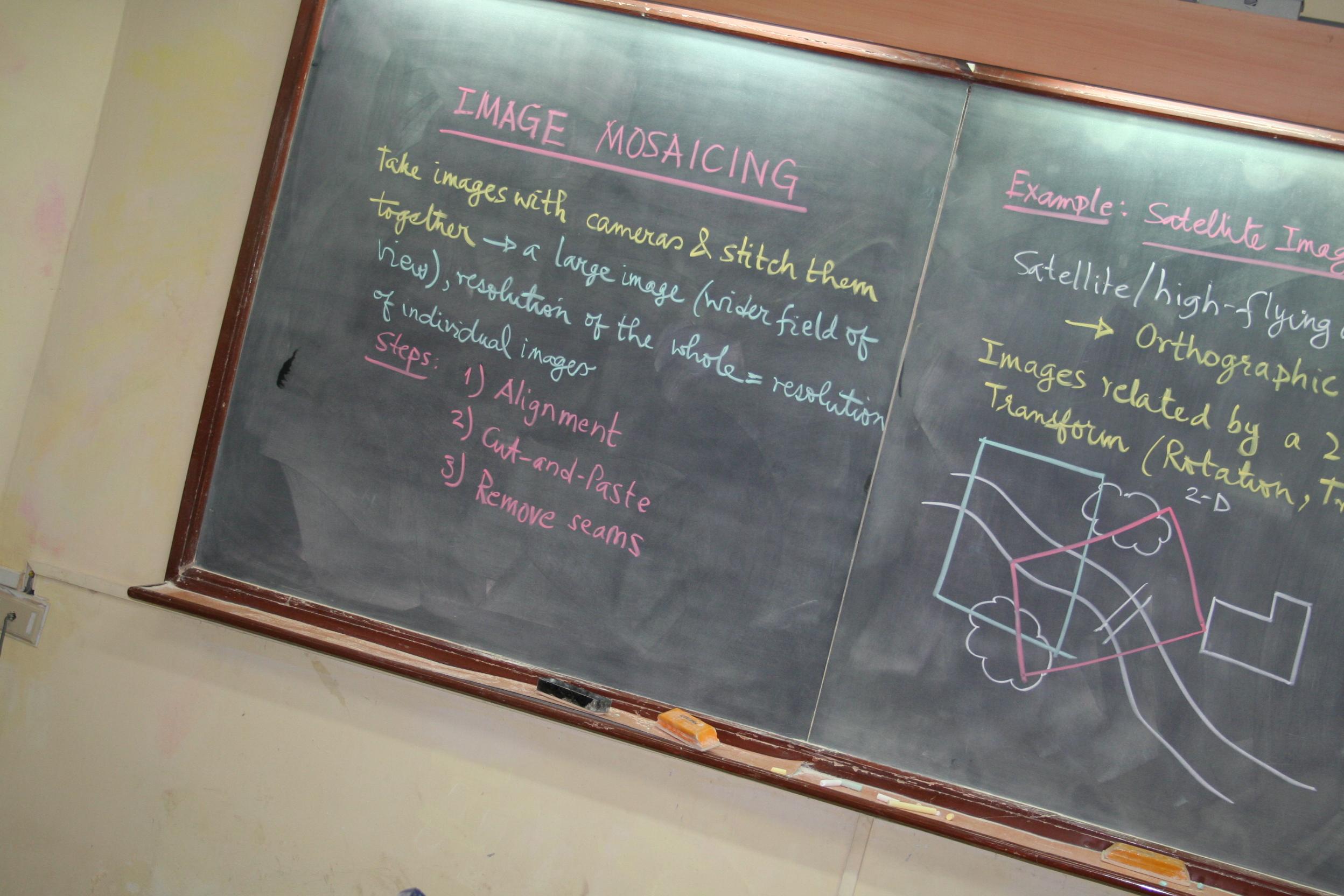
*M2.jpg*

**

*M3.jpg*

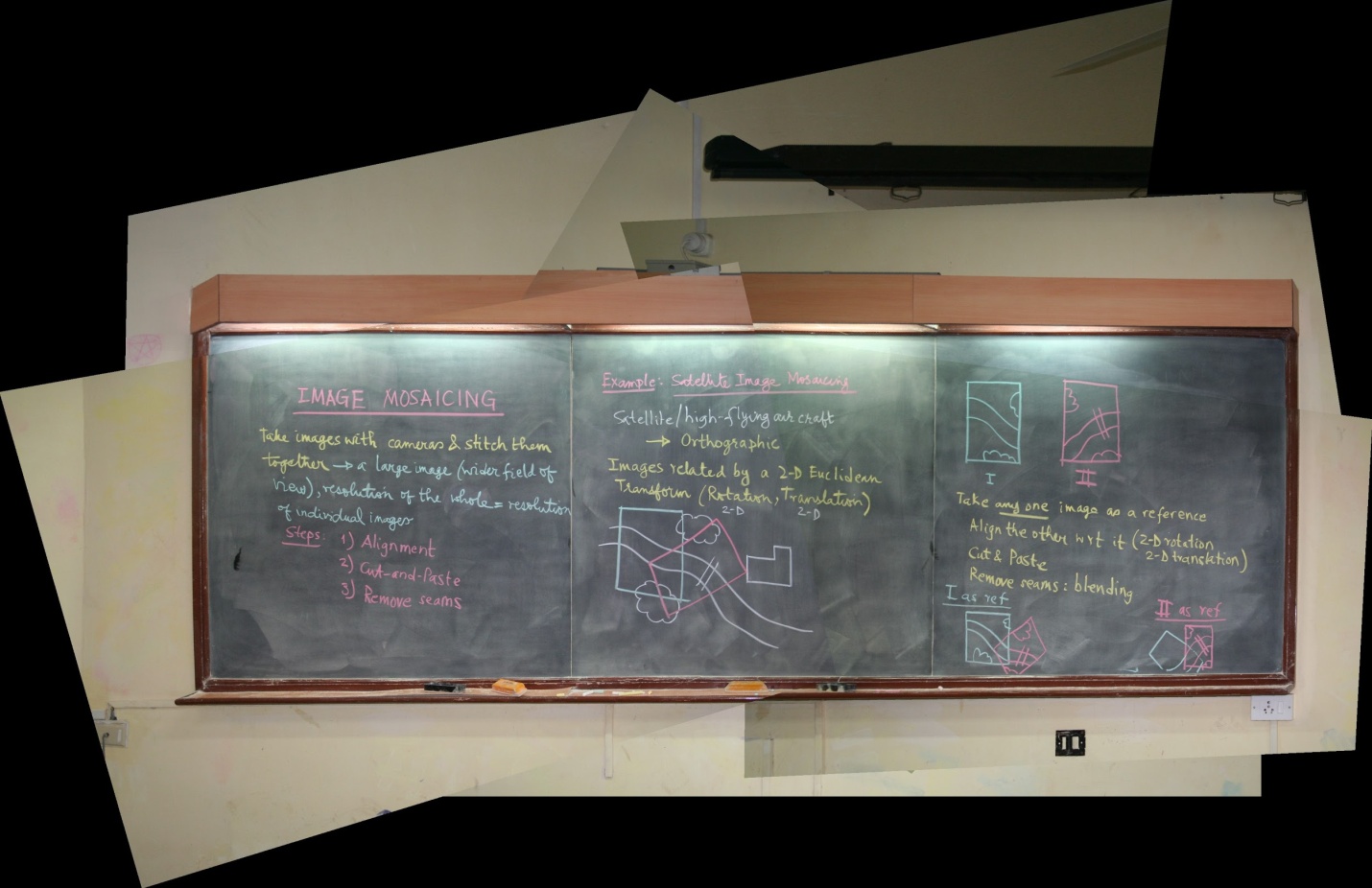
**

*M4.jpg*

**

*M5.jpg*

**OUTPUT IMAGE:**

******