ACV Project Report

Yasser Souri - 92204744 - ysouri@ce.sharif.edu

Code

Code is written using `python 2.7` programming language and `opencv 2.4.9` library.

Steps

In this section I will describe the steps necessary for adding the object virtually to the scene:

Camera Calibration: We use one of the chess boards for calibration. User must first select a M by N set of points (usually 8 by 7) in the chessboard. The first point is chosen as origin. And a

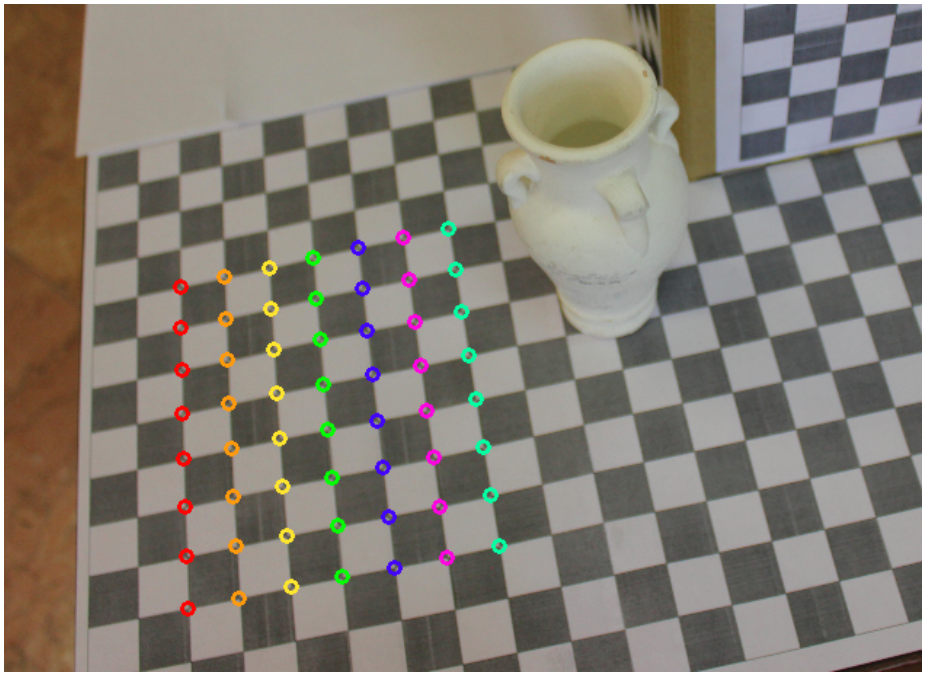

Figure 1: Example of points selected

Camera calibration is done using the opencv's functions. The output of camera calibration is all the intrinsic and extrinsic camera parameters, plus the distortion coefficients.

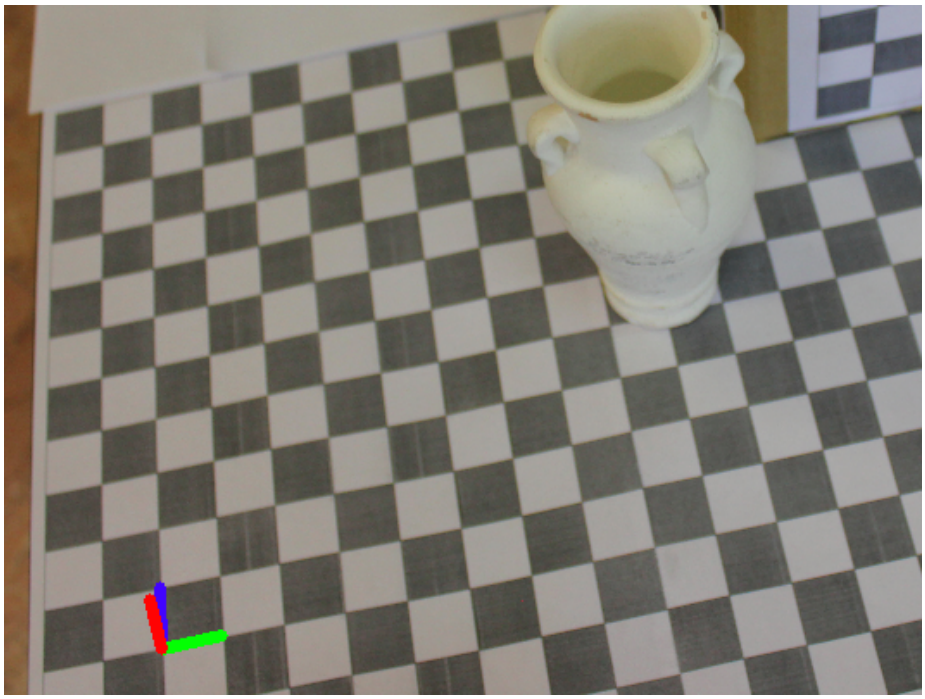

Figure 2: Example of world coordinate frame

Augmented Reality: We now have the projection matrix. All we need to do is to add the object to the scene. Here we use the values of *vertex*s as world points and by multiplying it with the projection matrix, we can get the image point for that point in the object.

Figure 3: Object Added as white pixels

Adding shadow: First we define a parametric line l that specifies the direction of light. Then for each object point loaded from *PLY* file, we pass a line with the same slope from it and find the intersection with the group plane ($z = 0$). The intersection of line and plane is the point of shadow. We then use the *faces* that was included with the *PLY* file to add small shadow to the body of the object. Each *face* is a set of 3 vertexes. With 3 vertexes, we can find the normal vector of a plane containing those 3 vertexes. By comparing this normal vector, with the slope vector of the light line, we can find a measure of how light should a pixel be. Then by smoothing these values we can get a better response

Automating the point selection

I have not been able to completely do this, but the point selection process is intelligent. It is sufficient to click near a point, not necessarily on it. The

Figure 4: Object added with shadows

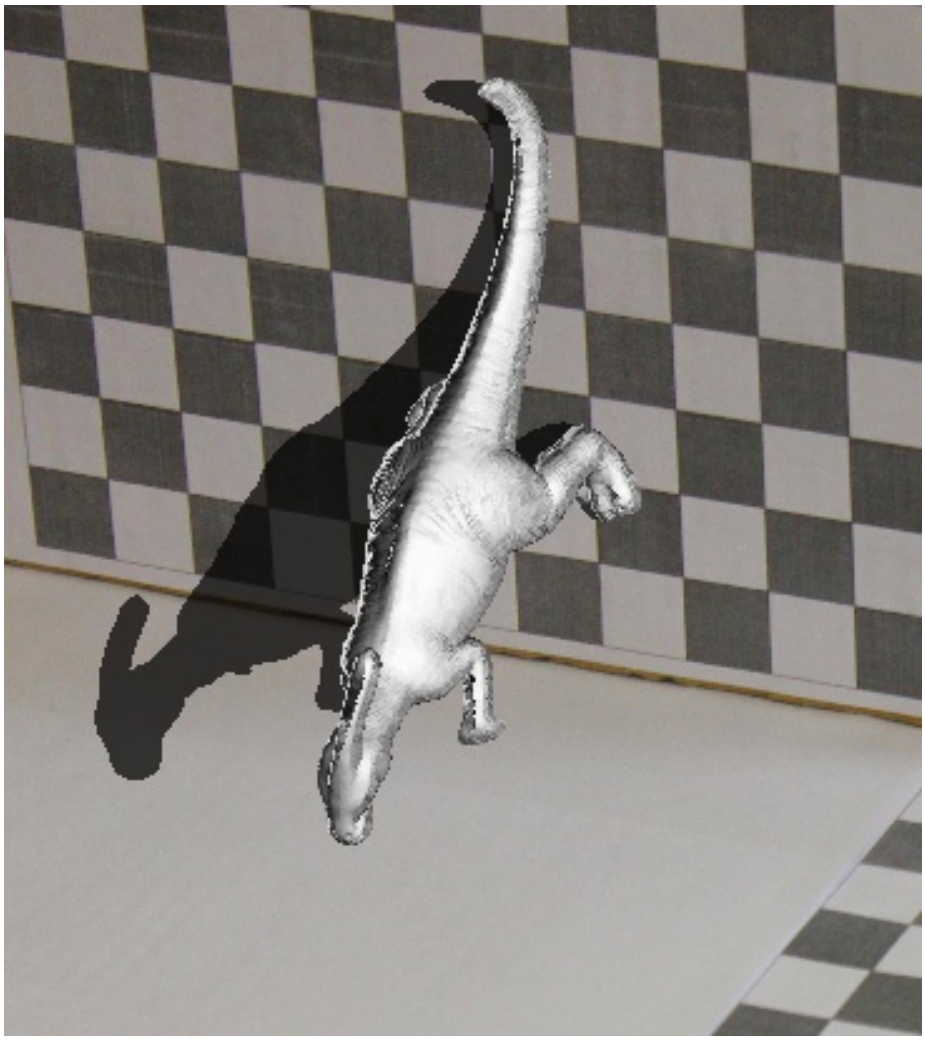

Figure 5: Another object added with shadows

program will automatically search nearby area in the image for corner like points. This is actually a lot of help.

Issues

Calibration: The calibration method that I have used is actually very sensitive to noise in points selected in the image. But in the dataset, the chessboard is sometimes not straight, and this causes problem for my method. Specially the chessboard on the ground is very wavy. This is the reason that I've chosen other surfaces some of the times.

Shadows: The reasoning about shadow in my program is very naive. For example one must detect if the path of light to a point in the world is blocked by other points. This currently is not implemented.

Runtime Efficiency: The algorithm is actually slow. For one thing this is implemented using `python`, but other than that, the algorithm itself might not be the fastest.

Final Results

Here are a collection of final results. As you might have notices, since the calibration method chosen needs between 30 to 60 points to produce meaningful results, I have not tested on the colored chessboard.

Dataset 3

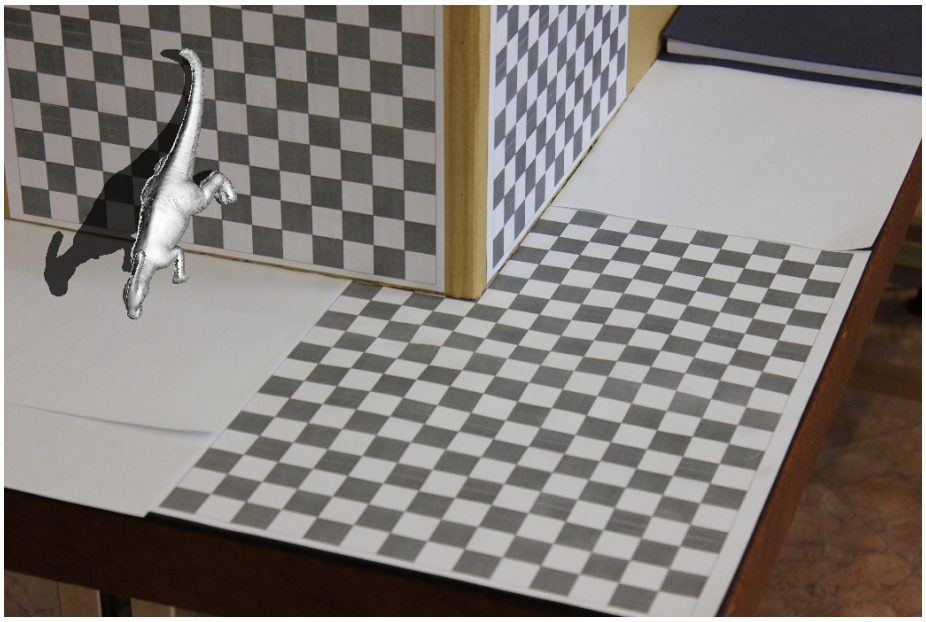

Figure 6: Dataset: 3, Image: 1

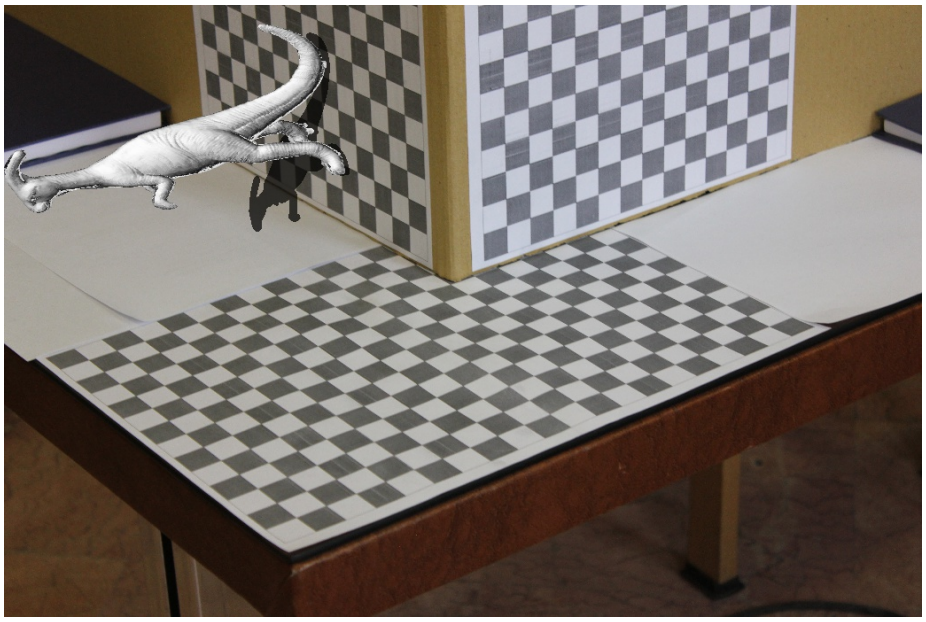

Figure 7: Dataset: 3, Image: 2

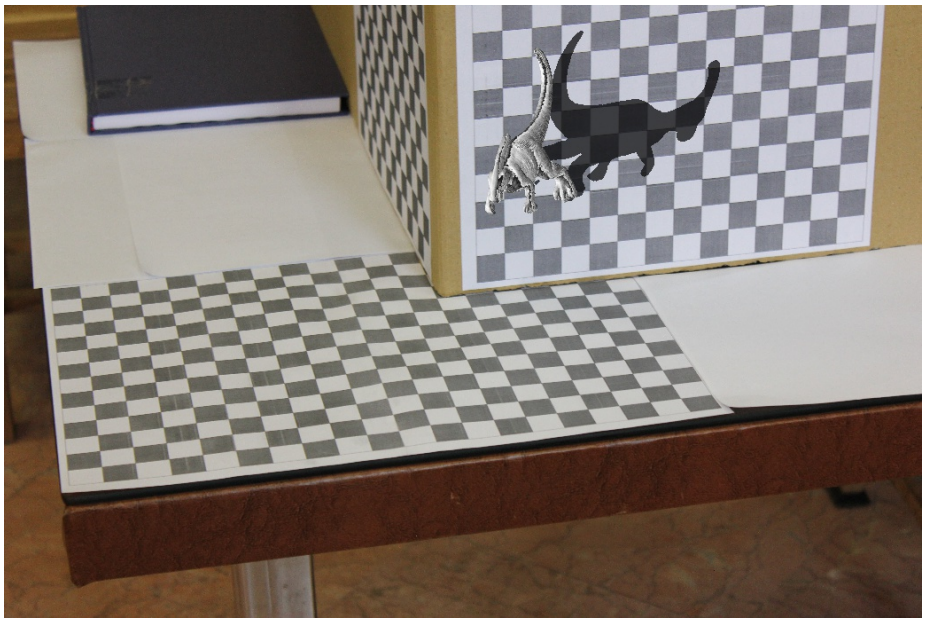

Figure 8: Dataset: 3, Image: 3

Dataset 4

Figure 9: Dataset: 4, Image: 1

Figure 10: Dataset: 4, Image: 2

Figure 11: Dataset: 4, Image: 3

Dataset 6

Figure 12: Dataset: 6, Image: 1

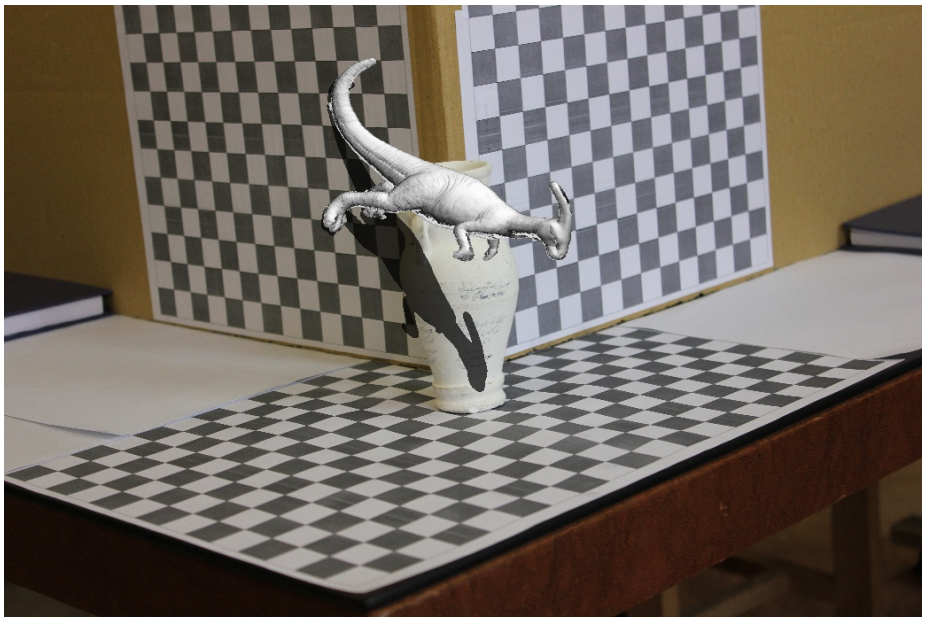

Figure 13: Dataset: 6, Image: 2

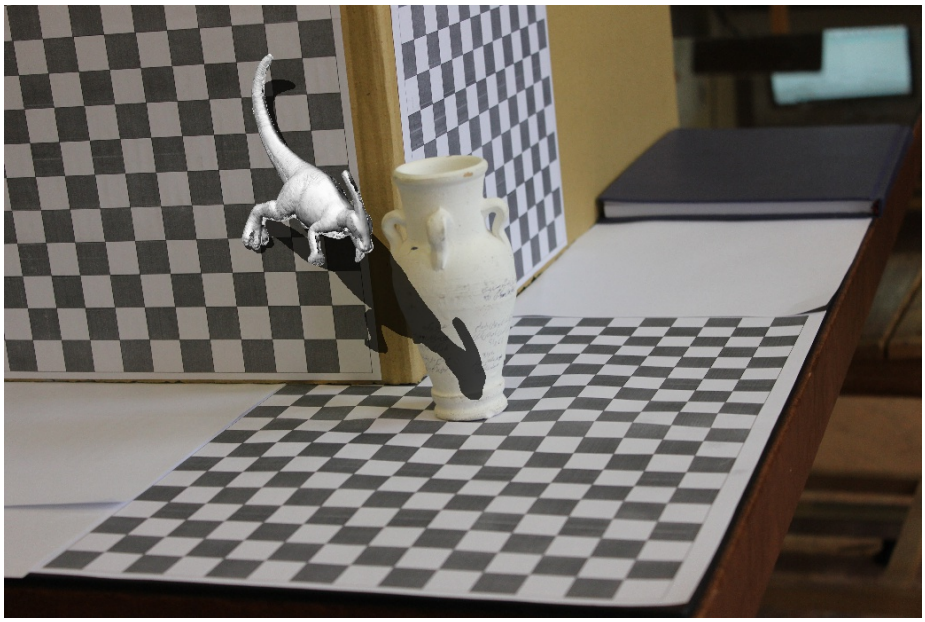

Figure 14: Dataset: 6, Image: 3

Dataset 7

Figure 15: Dataset: 7, Image: 1

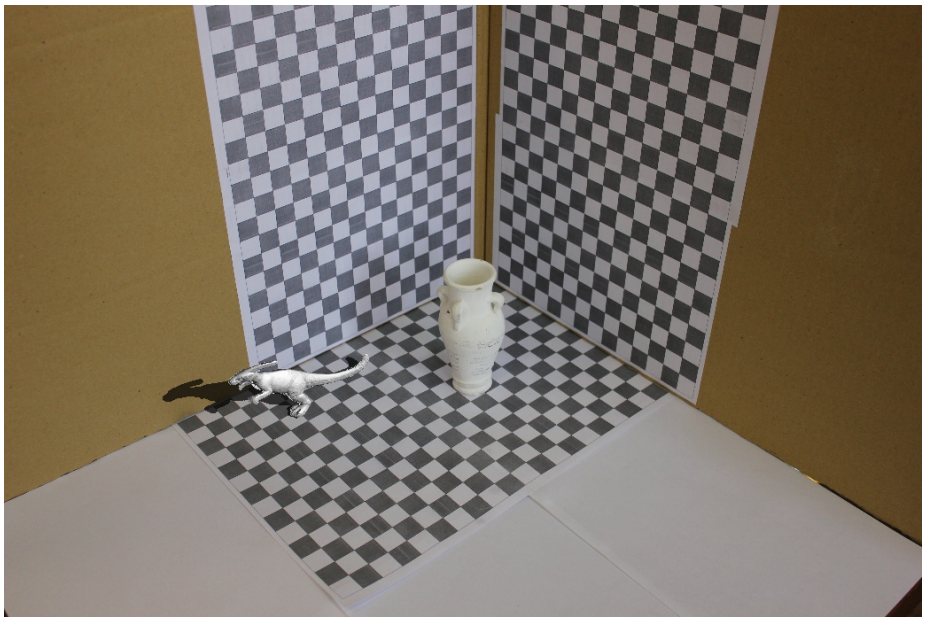

Figure 16: Dataset: 7, Image: 2

Figure 17: Dataset: 7, Image: 3