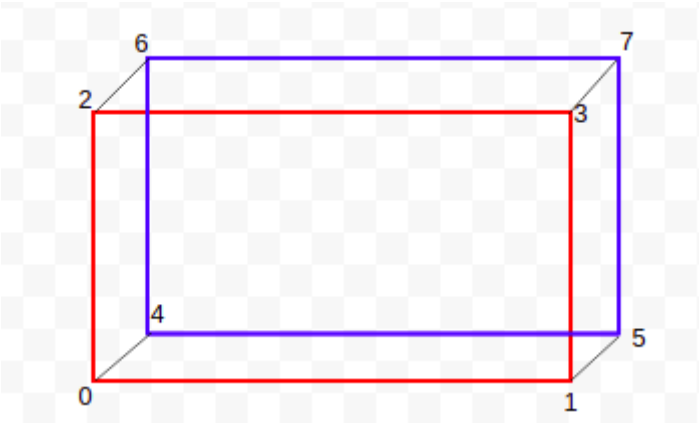
Reflection Essay

1. Introducing AR

Algorithm:

1. Read images and then calibrate camera to get instrinsics and extrinsics of the camera matrix (as done in previous task).
2. Next get the world coordinates of the 8 corners of the book.
   a. Fixed position of the book:
      With user input length, breadth and width of the book, fix the position of the book such that book binding is parallel to the horizontal. Given length=l, breadth=b, and width=w, the coordinates will be:
      Vertex 0: (0, 0, w), Vertex 1: (l, 0, w), Vertex 2: (0, b, w), Vertex 3: (l, b, w),
      Vertex 4: (0, 0, 0), Vertex 5: (l, 0, 0), Vertex 6: (0, b, 0), Vertex 7: (l, b, 0)
   b. Position of the book with given user input data:
      From user we would get the relative position of the center of the book (x,y) and orientation (theta); angle made by book binding with horizontal axis. By taking the co-ordinates axis of size 8*8, we would get the co-ordinates of the center of the book as x = 8*x/100, y = 8*y/100. Considering the initial coordinates (C=(x,y,z)) of center of the book be (0,0,0) and the co-ordinates of the corner according to length, breadth, and width of the book (which are also input from the user). The transformed co-ordinates of the book would be C' = R*(C,1) = R*(x,y,z,1) where R = $\begin{pmatrix} cos(\theta) & -sin(\theta) & 0 & x \\ sin(\theta) & cos(\theta) & 0 & y \\ 0 & 0 & 1 & 0 \end{pmatrix}$.
3. Calculate the respective pixel co-ordinates of the book using the camera matrix calculated. To calculate the pixel coordinates we use cv2.projectPoints(). It requires the rotation and translation parameters of the current view. If the view is one of the views which was used for calibration in step 1, then we already have it. Else we find the corresponding the rotation and translation parameters using cv2.solvePnP().
4. Using these pixel co-ordinates draw outline of the book on a given image (view).
5. ***Realism:*** To make it more real we stick texture of the front side with given images and left, right, top with white color. In order to do this we need to know two things. First, for a given view/orientation which side will be visible and which side isn't. Second we need to apply transformation on the texture images to place them, such that they match position and orientation.

   Let us consider cuboid shown below. Here 0123 represents front face and 4567 represents back face, which is sticked to the wall. The front face will be visible until either of the parallel lines doesn't coincide. A particular face (let say 2367) other than front face will be visible when the line which is common to the back face (67) and the line (01) parallel to the common edge (23) to the front face lie on the opposite side of the common edge and vice-versa. In the given diagram, side 0246 will not be visible as the lines 46 and 13 lies on the same side of the common edge 02. Whereas, side 1357 will be visible as the lines 57 and 02 lies on the opposite side of the common edge 13.



To check of the lines are on the same side or the opposite side of the common edge, we check if one of the corresponding vertices lie on the same side of the edge or not. To check if a two points lie on the same side of a line or not, we use the following. Consider the equation of the line in question be l:ax+by+x=0. Let the two points be A(x1,y1) and B(x2,y2). Let f(x,y) = ax+by+c. If the points A,B lie on the same side, then f(x1,y1)*f(x2,y2) >= 0 (same sign) else it will be < 0. Using this we can check if point A,B lie on the same side or not.

Once we know which side will be visible and which is not, we need to transform texture images such that they fit in the border of the side correctly.

Q1.1 Have the choices of views been made so that all the surfaces (except the back-cover) visible in at least one view?
Solution: Yes, all 5 surfaces are visible in atleast one of the views as you can see in the output images. View 1 is front view. View 2 is top and right side view. View 3 is downside view. View 4 is left side view.

Q1.2 How did you texture the book in the various views? If we were to provide you with a new view, or rotate the book by 90 degrees will your code be able to render the final result correctly?
Solution: Yes we would get the final result correctly for all different orientation of the book. We used perspective transform to get the correct texture orientation. We considered that the texture images are taken from a front on view. The we transform the front on view to the given view using cv2. getPerspectiveTransform() and cv2. warpPerspective(). We used the 4 corner points as the 4 correspondences.

```
src_pts = [[0, 0], [0, 480], [640, 0], [640, 480]]
src_pts = np.array(src_pts, dtype=np.float32)

matrix = cv2.getPerspectiveTransform(src_pts, dst_pts)
dest = cv2.warpPerspective(src, matrix, (1280, 960))
```

Outputs:


HARRY
POTTER


HARRY
POTTER
J.K. ROWLING


HARRY
POTTER
HARRY POTTER
J.K. ROWLING

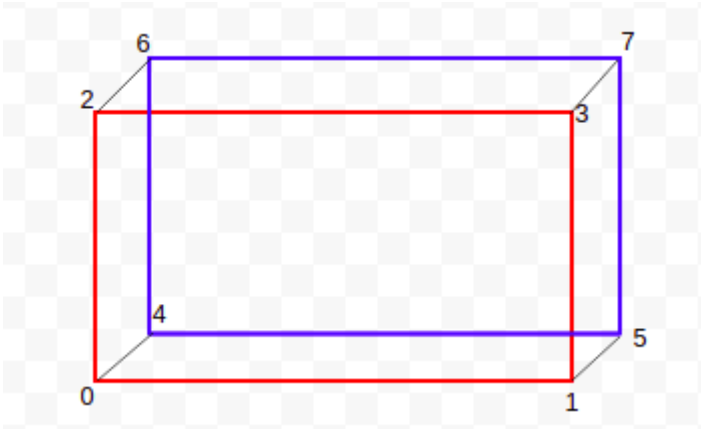

HARRY
POTTER


HARRY
POTTER


HARRY
POTTER

2. Mirror Mirror on the Wall.

2.1 Synthetic Mirror:



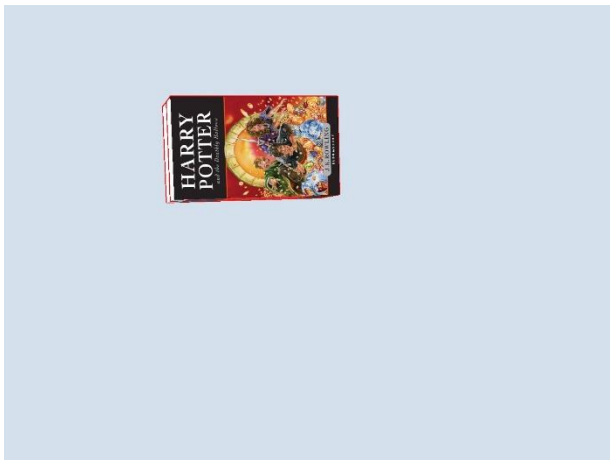
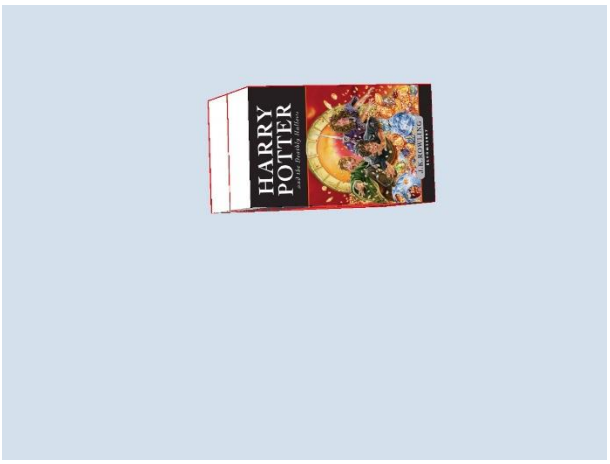
To find the reflection of the book, we note that the world coordinates of the bottom side of the book will be the same (as it is resting on the mirror). For the top side, the z-coordinates will just be negative of the z coordinates. For example the coordinate of the vertex 2 is (0, b, w). Its reflection will have a coordinate (0, b, -w). This is because the mirror is a flat mirror hence the distance of the object from the mirror and the distance of the reflection from the mirror is the same but they are on the opposite sides of the mirror. Using this, we find the world coordinates of the reflection of the book and then use the same logic as in part 1 to draw the book edges, determine the visible sides and draw the textures. For the synthetic part we create a synthetic image background and use the views of the previous task.

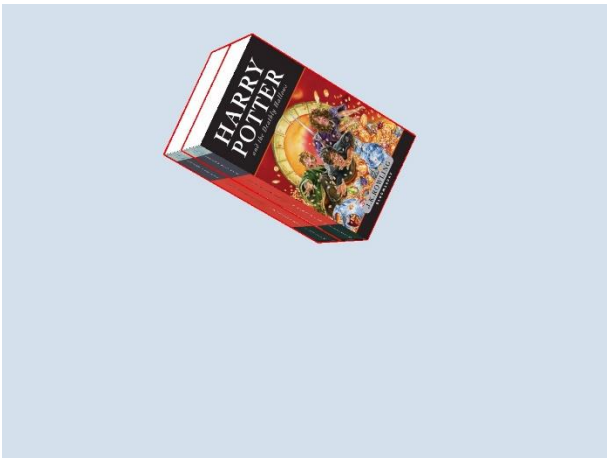
2.2 Real Mirror:

For the real part, we use cv2.solvePnP() to get the rotation and translation values and then project the world coordinate points to get the pixel points. Rest is same as synthetic case.

Outputs:

Synthetic Mirror:



Real Mirror: