Tutorial: 3D to 2D transform

In this tutorial we'll go through an example of a 3D-to-2D transform.

Context

A camera is mounted on a robot vehicle. The center of the camera is located at (X,Y,Z) = (0,+1 ,+2) in vehicle coordinates, and the camera is tilted downwards by 30 degrees

The vehicle is located at (X,Y,Z) = (+4,-4,+1) in world coordinates, with a rotation angle of 30 degrees with respect to the world.

The camera observes four points on the ground, with world coordinates at (0,0,0), (1,0,0), (1,1,0), and (0,1,0).

Task

Generate a synthetic image of the points as seen from the camera, assuming the image is 200 pixels high by 300 pixels wide, with an effective focal length of 300 pixels.

Process

1. Determine the transformation from vehicle to world
    1. Define ax, ay, and az, the angles of rotation of the vehicle with respect to the world.
    2. Plug these values into the rotation matrix equations to calculate Rx, Ry, and Rz.
    3. Multiply Rz * Ry * Rx to get the full rotation matrix.
    4. Define tv_w, the translation vector for the vehicle within the world.
    5. Combine R_v_w and tv_w with an additional row [0 0 0 1] to produce your 4x4 transformation matrix H_v_w.
2. Determine the transformation from camera to vehicle
    1. Follow the same process as in step one, but make sure to use the translation and rotation of the camera with respect to the vehicle rather than the vehicle with respect to the world. This will produce transformation matrix H_c_v.
3. Multiply these matrices to produce transformation matrix H_c_w, from camera to world. Make sure you are multiplying in the appropriate order: matching terms should go from top-left to bottom-right.

4. Invert this matrix (e.g. in numpy using np.linalg.inv()) to get the transformation matrix H_w_c, from world to camera.
5. Pull out the first three rows. This is your extrinsic parameter matrix Mext.
6. Define your intrinsic parameter matrix K.
7. Define a matrix containing your world points, P_w, where each column represents a single point in homogeneous coordinates.
8. Pre-multiply your points by your extrinsic parameter matrix, and then premultiply those results by your intrinsic parameter matrix to project the coordinates to image points pimg. Then divide each row of this matrix by the third row to make sure they are appropriately normalized.
9. Create your image!
    1. Create an image matrix of the appropriate size using np.zeros()
    2. Iterate over your array of image points. You can get the number of columns by using ping.shape[1].
        1. For each image point, pull out and round the column and row (e.g., c=int(round(pimg[0,i])) for that point.
        2. Set the cell in your image matrix at that row and column equal to 1.
    3. Show your image using either OpenCV's imshow, or matplotlib's imshow!