All conversions

- left-Hand to Right-Hand
- img2pointCloud
- 3D to 2D
    - Projection
    - Rasterization

1. Left-Hand to Right-Hand Coordinate System[Rest of the world]

- X,Y,Z -> X, -Y, Z
- roll, pitch, yaw -> roll, -pitch, -yaw

2. img2pointCloud

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

2D Image Coordinates | Intrinsic properties (Optical Centre, scaling) | Extrinsic properties (Camera Rotation and translation) | 3D World Coordinates

Maths:

`XYZ = Inv(K) * [u,v,1]`

`X,Y,Z -> Z, X, Y`

The code

```
def depth_to_lidar(self, normalized_depth, w, h, K, max_depth=0.9):
    """
    Convert an image containing CARLA encoded depth-map to a 2D array
containing
    the 3D position (relative to the camera) of each pixel and its
corresponding
    RGB color of an array.
    "max_depth" is used to omit the points that are far enough.
    """
    far = 1000.0  # max depth in meters.
    w,h,K = int(w), int(h), np.array(K)
    pixel_length = w*h
    u_coord = np.matlib.repmat(np.r_[w-1:-1:-1],
                   h, 1).reshape(pixel_length)
    v_coord = np.matlib.repmat(np.c_[h-1:-1:-1],
                   1, w).reshape(pixel_length)
    normalized_depth = np.reshape(normalized_depth, pixel_length)
    # Search for pixels where the depth is greater than max_depth to
    # Make them = 0 to preserve the shape
```

```
    max_depth_indexes = np.where(normalized_depth > max_depth)
    normalized_depth[max_depth_indexes] = 0
    u_coord[max_depth_indexes] = 0
    v_coord[max_depth_indexes] = 0
    depth_np_1d = normalized_depth *far

    # p2d = [u,v,1]
    p2d = np.array([u_coord, v_coord, np.ones_like(u_coord)])

    # P = [X,Y,Z] # Pixel Space to Camera space
    p3d = np.dot(np.linalg.inv(K), p2d)
    p3d *= depth_np_1d

    lidar_np_3d = np.transpose(p3d)
    py,pz,px = lidar_np_3d[:, 0], lidar_np_3d[:, 1], lidar_np_3d[:, 2]

    lidar_np_3d = np.vstack((px,py,pz))
    depth_np_1d = np.reshape(depth_np_1d, (h, w))

    # header = laspy.LasHeader(point_format=0, version="1.2")
    # las = laspy.LasData(header)
    # las.x, las.y, las.z = px, py, pz
    return lidar_np_3d, depth_np_1d
```

3. 3D to 2D

- Projection