Tutorial 3
Matrix operation on points

If you want to perform operation $R_1, \ldots R_n$ on point
\[
\begin{bmatrix}
w x \\
w y \\
w 
\end{bmatrix}
\]

\[
R_n \cdot R_{n_1} \cdots R_1 \cdot \begin{bmatrix}
w x \\
w y \\
w 
\end{bmatrix}
\]

Operation of operations is reverse to the order of multiplying matrices.
Translation matrix

\[
\begin{pmatrix}
1 & 0 & t_x \\
0 & 1 & t_y \\
0 & 0 & 1
\end{pmatrix}
\begin{bmatrix}
w \cdot x \\
w \cdot y \\
w
\end{bmatrix} =
\begin{bmatrix}
w(x + t_x) \\
w(y + t_y) \\
w
\end{bmatrix} = w \cdot \left(
\begin{bmatrix}
x \\
y \\
1
\end{bmatrix} +
\begin{bmatrix}
t_x \\
t_y \\
1
\end{bmatrix}
\right)
\]
Rotation matrix

\[
\begin{pmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{pmatrix}
\]

Rotation is around \((0, 0)\) and is counter clockwise
Excercise

Given some polygon, rotate it $45^\circ$ clockwise around it's center using translation and rotation matrices

In [4]:
points = [
    (5, 6),
    (5, 7),
    (6, 8),
    (6, 6)]
In [5]:

draw_poly_mat(np.array(points).transpose(), figsize=(7, 7))

/usr/lib/python3/dist-packages/matplotlib/axes/_base.py:3215: MatplotlibDeprecationWarning:
The `xmin` argument was deprecated in Matplotlib 3.0 and will be removed in
3.2. Use `left` instead.
    alternative='`left`', obj_type='argument')
/usr/lib/python3/dist-packages/matplotlib/axes/_base.py:3221: MatplotlibDeprecationWarning:
The `xmax` argument was deprecated in Matplotlib 3.0 and will be removed in
3.2. Use `right` instead.
    alternative='`right`', obj_type='argument')
/usr/lib/python3/dist-packages/matplotlib/axes/_base.py:3604: MatplotlibDeprecationWarning:
The `ymin` argument was deprecated in Matplotlib 3.0 and will be removed in
3.2. Use `bottom` instead.
    alternative='`bottom`', obj_type='argument')
/usr/lib/python3/dist-packages/matplotlib/axes/_base.py:3610: MatplotlibDeprecationWarning:
The `ymax` argument was deprecated in Matplotlib 3.0 and will be removed in
3.2. Use `top` instead.
    alternative='`top`', obj_type='argument')
\[ H \cdot \begin{pmatrix} x_1 & x_2 & \ldots \end{pmatrix} = \begin{pmatrix} H \cdot \begin{pmatrix} x_1 \end{pmatrix} & H \cdot \begin{pmatrix} x_2 \end{pmatrix} & \ldots \end{pmatrix} \]
Points to matrix of column vectors

In [6]:
np_points = np.array(points).transpose()
print(np_points)

[[5 5 6 6]
 [6 7 8 6]]
Transform to homogenius coordinates

In [7]:
np_points_h = np.vstack([np_points,
                        np.ones((1, len(points)))]
print(np_points_h)

[[5. 5. 6. 6.]
 [6. 7. 8. 6.]
 [1. 1. 1. 1.]]
Calculating center and angle

```
In [8]:
center = np.sum(np_points, axis=1) / len(points)
theta = np.deg2rad(45)
print(center)

[5.5 6.75]
```
In [9]: m_1 = np.array([
        [1, 0, -center[0]],
        [0, 1, -center[1]],
        [0, 0, 1]
    ])  
   m_1

Out[9]: array([[ 1. ,  0. , -5.5 ],
               [ 0. ,  1. , -6.75],
               [ 0. ,  0. ,  1. ]])
In [10]:
    m_2 = np.array([
    [np.cos(-theta), -np.sin(-theta), 0],
    [np.sin(-theta), np.cos(-theta), 0],
    [0, 0, 1]
    ])

m_2

Out[10]:
array([[ 0.70710678,  0.70710678,  0.        ],
       [-0.70710678,  0.70710678,  0.        ],
       [ 0.        ,  0.        ,  1.        ]])
In [11]:
   m_3 = np.array([
    [1, 0, center[0]],
    [0, 1, center[1]],
    [0, 0, 1]
   ])
   m_3

Out[11]:
   array([[ 1. ,  0. ,  5.5 ],
    [ 0. ,  1. ,  6.75],
    [ 0. ,  0. ,  1. ]])
In [12]:

```python
resh = m_3 @ m_2 @ m_1 @ np_points_h
rese = resh[:, :, :] / resh[2, :, :]
draw_poly_mat(rese, figsize=(7, 7))
```
World point / image point relation

- What is the geometric shape of all image points corresponding to a world point?
- What is the geometric shape of all world points corresponding to an image point?
World point / image point relation

- What is the geometric shape of all image points corresponding to a world point? A point
- What is the geometric shape of all world points corresponding to an image point? A line
Camera calibration
World point to image point \[
\begin{bmatrix}
w \cdot x_i \\
w \cdot y_i \\
w
\end{bmatrix}
\underbrace{K \cdot}_{M}
\begin{bmatrix}
R & T
\end{bmatrix}
\begin{bmatrix}
x_w \\
y_w \\
z_w \\
1
\end{bmatrix}
\]

- **M** - The camera matrix
- **T** - Position of the world center relative to camera center
- **R** - Rotation of the world center relative to camera axis
- **K** - Intrinsic camera matrix
Intrinsic camera matrix:

\[ K = \begin{pmatrix}
  f_x & 0 & c_x & 0 \\
  0 & f_y & c_y & 0 \\
  0 & 0 & 1 & 0
\end{pmatrix} \]
Calibration workflow

1. Take an image of an object you know its shape and from a known axis and identify the image points
2. Create pairs of world points to image points
3. Use opencv algorithms to get camera matrix
stage 1

We use an object we can identify automatically in our program.

We use a chessboard because OpenCV has a method to automatically identify its corners in the image.
```python
In [14]:
chess_img = cv2.imread('chess.jpg', cv2.IMREAD_GRAYSCALE)
imshow(chess_img, cmap='gray', figure=figure(figsize=(10,10)))
```

```
Out[14]:
<matplotlib.image.AxesImage at 0x7f71f278cba8>
```
In [16]:
# The found variable is a boolean indicating if the corners were detected
# The corners variables stores the found corners
# Note that the corners variable is a list of 2D points in the image
found, corners = cv2.findChessboardCorners(chess_img, pattern_size)
```python
In [17]: imshow(cv2.drawChessboardCorners(cv2.cvtColor(chess_img, cv2.COLOR_GRAY2RGB), pattern_size, corners, found),
          figure=figure(figsize=(10, 10)))

Out[17]: <matplotlib.image.AxesImage at 0x7f71f26fcbe0>
```
stage 2

We have found our image point (2D) in Stage 1 and now we want to find a corresponding world (3D) point for each image point.

In order to talk about world points, we need to define the world axis.

To make our life easy, we choose an axis such that the $x, y$ plane lays on top of the chessboard and the origin lays at the upper left corner of the board.

To make our life even easier, we choose the unit of measure to be a chessboard piece. The next cells shows two methods to construct the points
through python
In [18]:
from itertools import product
xs, ys = pattern_size
world_points = np.array([(x, y, 0)
                         for y, x in product(range(ys), range(xs))],
                         dtype=np.float32)
print(world_points)

[[0. 0. 0.]
 [1. 0. 0.]
 [2. 0. 0.]
 [3. 0. 0.]
 [4. 0. 0.]
 [5. 0. 0.]
 [0. 1. 0.]
 [1. 1. 0.]
 [2. 1. 0.]
 [3. 1. 0.]
 [4. 1. 0.]
 [5. 1. 0.]
 [0. 2. 0.]
 [1. 2. 0.]
 [2. 2. 0.]
 [3. 2. 0.]
 [4. 2. 0.]
 [5. 2. 0.]
 [0. 3. 0.]
 [1. 3. 0.]
 [2. 3. 0.]
 [3. 3. 0.]
 [4. 3. 0.]
 [5. 3. 0.]
 [0. 4. 0.]
 [1. 4. 0.]
 [2. 4. 0.]
 [3. 4. 0.]
 [4. 4. 0.]
 [5. 4. 0.]
]
Through Numpy
In [19]:
xs, ys = pattern_size
xx, yy = np.meshgrid(np.arange(xs),
                    np.arange(ys))
np.vstack([xx.flatten(),
            yy.flatten(),
            np.zeros(xs * ys)]).transpose()

Out[19]:
array([[0., 0., 0.],
       [1., 0., 0.],
       [2., 0., 0.],
       [3., 0., 0.],
       [4., 0., 0.],
       [5., 0., 0.],
       [0., 1., 0.],
       [1., 1., 0.],
       [2., 1., 0.],
       [3., 1., 0.],
       [4., 1., 0.],
       [5., 1., 0.],
       [0., 2., 0.],
       [1., 2., 0.],
       [2., 2., 0.],
       [3., 2., 0.],
       [4., 2., 0.],
       [5., 2., 0.],
       [0., 3., 0.],
       [1., 3., 0.],
       [2., 3., 0.],
       [3., 3., 0.],
       [4., 3., 0.],
       [5., 3., 0.],
       [0., 4., 0.],
       [1., 4., 0.],
       [2., 4., 0.],
       [3., 4., 0.],
       [4., 4., 0.],
       [5., 4., 0.]])
Stage 3

In this stage we obtain the camera matrix using our (image - world) point pairs we obtained at previous stages. In your homework you will use the Opencv documentation to analyze the results.

**Note** - In this example we used a single image for calibration, but in your homework you will need to use multiple images. For this, you must repeat stages 1 and 2 for each image.

```
In [20]: retval, cameraMatrix, distCoeffs, rvecs, tvecs = cv2.calibrateCamera(
            objectPoints=[world_points],
            imagePoints=[corners],
            imageSize=chess_img.shape[:2],
            cameraMatrix=None,
            distCoeffs=None)

print(cameraMatrix)
```

```
[[ 417.44068475 0. 226.98026902]
 [ 0. 436.23345401 164.17134843]
 [ 0. 0. 1. ]]
```