Assignment 2 - Epipolar Geometry and 3D Reconstruction

First name: Brian

Last name: Schweigler

Matriculation number: 16-102-071

In [1]:
```
%load_ext autoreload
%autoreload 2
%matplotlib inline
import os

import numpy as np
from PIL import Image
import ipyvolume as ipv
import matplotlib.pyplot as plt
from scipy.spatial.transform import Rotation

from utils import decompose_essential_matrix, infer_3d, ransac
```

Part II: 3D Model Reconstruction

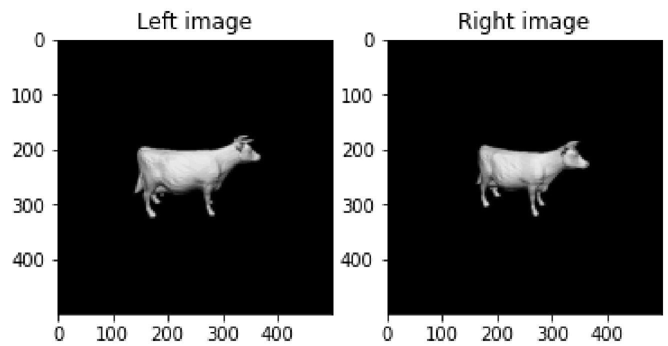
Load matched points

We provide a synthetic pair of images where noisy correspondences are known.

In [2]:
```
left = np.array(Image.open(os.path.join('MatchedPoints', 'left.jpg')), dtype=np.float32).mean(2) /
right = np.array(Image.open(os.path.join('MatchedPoints', 'right.jpg')), dtype=np.float32).mean(2)
```

In [3]:
```
plt.subplot(1, 2, 1)
plt.imshow(left, cmap='gray')
plt.title('Left image')
plt.subplot(1, 2, 2)
plt.imshow(right, cmap='gray')
plt.title('Right image')
```

Out[3]:
```
Text(0.5, 1.0, 'Right image')
```



In [4]:
```
ilias_username = 's.b.marvin'
_A = np.loadtxt(f'MatchedPoints/Matched_Points_{ilias_username}.txt')
```

In [5]:
```
_M, _N = _A.shape
leftPoints = np.concatenate((_A[:, 2:4].T, np.ones((1, _M))), axis=0)
rightPoints = np.concatenate((_A[:, 0:2].T, np.ones((1, _M))), axis=0)
```

Calibration matrix and focal length from the given file

In [6]:
```
fl = 4
_K = np.array([
    [-83.33333, 0.00000, 250.00000],
    [0.00000, -83.33333, 250.00000],
    [0.00000, 0.00000, 1.00000],
])

_I = _K.copy()

_I[0, 0] *= fl
_I[1, 1] *= fl
```

Estimate Essential matrix E from F with RANSAC

In [7]:
```
good_threshold = 0.925
_F, inliers = ransac(leftPoints, rightPoints, good_threshold)
print('Num outliers', leftPoints.shape[1] - inliers.sum())
assert np.linalg.matrix_rank(_F) == 2
print('Estimated fundamental matrix: ')
print(_F)

# Estimate essential matrix E from F
_E = np.matmul(np.matmul(_I.T, _F), _I)

print('Estimated essential matrix: ')
print(_E)
```

```
Num outliers 2
Estimated fundamental matrix:
[[-5.77322682e-08 -1.73558176e-05  6.72768961e-03]
 [-2.89167439e-07  2.42527451e-06  1.63815808e-02]
 [ 7.12388194e-05 -1.22477094e-02 -1.77811511e+00]]
Estimated essential matrix:
[[-6.41469595e-03 -1.92842403e+00 -7.91434013e-01]
 [-3.21297129e-02  2.69474924e-01 -5.63853564e+00]
 [ 5.16203558e-03  5.32678152e+00  2.44670751e-04]]
```

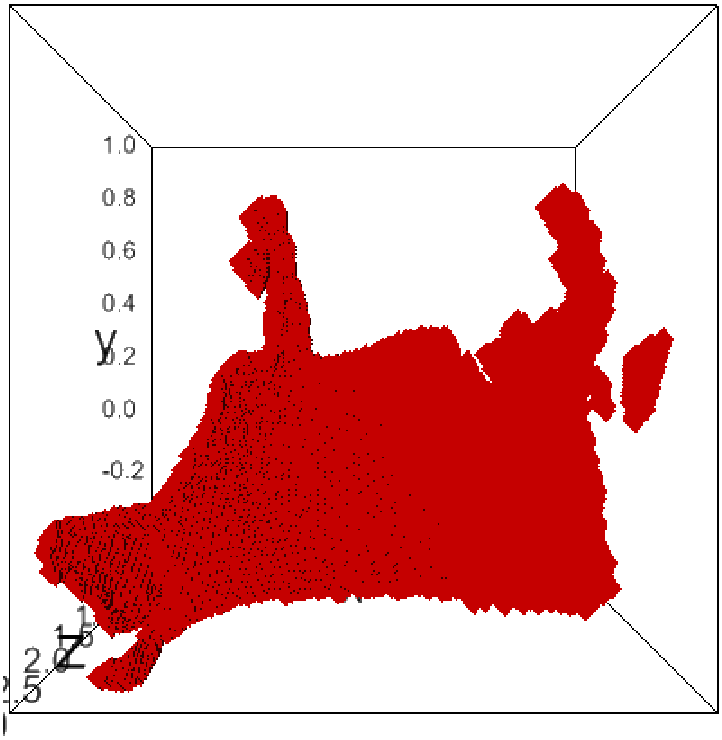
Compute rotation and translation between views

In [8]:
```
# Compute rotation and translation between views. Complete decomposeE
_Il = np.linalg.solve(_I, leftPoints[:, inliers])
_Ir = np.linalg.solve(_I, rightPoints[:, inliers])

_Pl, _Pr = decompose_essential_matrix(_E, _Il, _Ir)

print('Estimated translation: ')
print(_Pr[:, 3])
print('Estimated rotation: ')
print(_Pr[:, :3])
```

```
Estimated translation:
[ 0.93010197 -0.13053572  0.34332309]
Estimated rotation:
[[ 9.28267689e-01  1.38056418e-01 -3.45339722e-01]
 [-1.29300463e-01  9.90424332e-01  4.83842199e-02]
 [ 3.48712615e-01 -2.60921877e-04  9.37229664e-01]]
```

Estimate the 3D points

In [17]:
```
x3d = infer_3d(_Il, _Ir, _Pl, _Pr)
ipv.quickscatter(x=x3d[0, :], y=x3d[1, :], z=x3d[2, :])
```



Show Errors of Estimated Extrinsics

In [10]:

```python
# Right camera's rotation matrix
r_rot_mat = np.array([[0.92848, -0.12930, 0.34815],
                      [0.00000, 0.93744, 0.34815],
                      [-0.37139, -0.32325, 0.87039]])

# Right Camera's Translation vector (inverted x compared to world frame)
r_tr_mat = np.array([-2.0, 2.0, 5.0])

# Homogenous Transformation matrix (Right Camera > World)
HT_right_to_world = np.zeros((4, 4))
HT_right_to_world[:3, :3] = r_rot_mat
HT_right_to_world[:3, 3] = r_tr_mat
HT_right_to_world[3, 3] = 1

# Left camera's rotation matrix
l_rot_mat = np.array([[1.00000, 0.00000, 0.00000],
                      [0.00000, 0.92848, 0.37139],
                      [0.00000, -0.37139, 0.92848]])

# Left Camera's Translation vector (inverted x compared to world frame)
l_tr_mat = np.array([0.0, 2.0, 5.0])

# Homogenous Transformation matrix (World > Left Camera)
HT_w_to_left = np.zeros((4, 4))
HT_w_to_left[:3, :3] = l_rot_mat.T
HT_w_to_left[:3, 3] = np.matmul(- l_rot_mat.T, l_tr_mat)
HT_w_to_left[3, 3] = 1

# Homogenous Transformation matrix (Right Camera > Left Camera)
HT_right_to_left = np.matmul(HT_w_to_left, HT_right_to_world)

# Pr is inverse of this matrix, thus invert HT_r_to_l
HT_left_to_right = np.zeros((4, 4))
HT_left_to_right[:3, :3] = HT_right_to_left[:3, :3].T
HT_left_to_right[:3, 3] = np.matmul(- HT_right_to_left[:3, :3].T, HT_right_to_left[:3, 3])
HT_left_to_right[3, 3] = 1

# Get Rotation Matrix and Translation Vector
l_rot_mat_calculated = HT_left_to_right[:3, :3]
l_tr_mat_calculated = HT_left_to_right[:3, 3]
l_rot_mat_approximated = _Pr[:3, :3]
l_tr_mat_approximated = _Pr[:3, 3]

# Normalize Translation Vector
l_tr_mat_normalized = l_tr_mat_calculated / np.linalg.norm(l_tr_mat_calculated)

# Euler Angles
a_spatial_rot_mat = Rotation.from_matrix(l_rot_mat_approximated)
c_spatial_rot_mat = Rotation.from_matrix(l_rot_mat_calculated)

approximated_euler_angles = a_spatial_rot_mat.as_euler("xyz", degrees=True)
calculated_euler_angles = c_spatial_rot_mat.as_euler("xyz", degrees=True)

# Calculate Rotation & Translation Error
translation_error = np.sum(np.abs(l_tr_mat_approximated - l_tr_mat_normalized))
rotation_error = np.sum(np.abs(approximated_euler_angles - calculated_euler_angles))
```

Results

Rotation Angles

In [11]:
```
print('Approximated Rotation Angles: {}'.format(approximated_euler_angles))
print('Calculated Rotation Angles: {}'.format(calculated_euler_angles))
```

```
Approximated Rotation Angles: [-1.59509697e-02 -2.04085931e+01 -7.92983306e+00]
Calculated Rotation Angles: [-2.42121284e-04 -2.03743564e+01 -7.92801481e+00]
```

Translation Vectors

In [12]:
```
print('Approximated Translation Vector: {}'.format(l_tr_mat_approximated))
print('Calculated Translation Vector: {}'.format(l_tr_mat_normalized))
```

```
Approximated Translation Vector: [ 0.93010197 -0.13053572  0.34332309]
Calculated Translation Vector: [ 0.92847906 -0.12929987  0.34814965]
```

l^1-norm errors

In [13]:
```
print('Translation Error (L1-Norm): {}'.format(translation_error))
print('Rotation Error (L1-Norm): {}'.format(rotation_error))
```

```
Translation Error (L1-Norm): 0.007685324445127983
Rotation Error (L1-Norm): 0.05176386731421853
```