Assignment 1 (part I): Line Fitting and other "stuff"

Problem 1

Prove that affine transformations map lines onto lines. For this, take an arbitrary line in R^2 and show that an arbitrary affine transofrmation maps it onto a set of points that satisfy a line equation. Use linear algebraic equations for lines, i.e. $x^T v = c$ where v is a 2-vector (normal to the line), c is a scalar (v, c are line parameters), and x is a 2-vector corresponding to an arbitrary point on the line. Your proof should use only linear algebraic equations.

HINT: for inspiration, check out the proof for homographies on slide 33, topic 4.

Solution:

Take an arbitrary line in R^2, $y = ax + b$. The points on this line are $p = (x, y)$. We can define a vector $\vec{p} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ to represet all points in homogeneous coordinates for convenience. The equation of the line can be rewritten in matrix notation as follows:

$$\begin{bmatrix} -a & 1 & -b \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0$$
$$\vec{c}^T \cdot \vec{p} = 0$$

An affine transformation matrix has the following form:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$$

where a, b, c, d, e, f are scalars. Let this matrix be called A.

It has a determinant as follows:

$$det(A) = det\left(\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}\right) \qquad = (1) \cdot |ae - bd|$$

As long as $ae \neq bd$ the determinant of A, $det(A) \neq 0$ and the inverse matrix A^{-1} is defined.

We know that an affine transformation takes points (x, y) and converts them to new points (x', y'). This can be represented using matrix multiplication and their homogenous coordinate representations.

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix}$$
$$A \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix}$$

These new points $p' = (\frac{x'}{w'}, \frac{y'}{w'})$ form a line like they did for the points p, $\frac{y'}{w'} = a'\frac{x'}{w'} + b'$ or $y' = a'x' + b'w'$. In matrix notation and using homogenous coordinates $\vec{p} = \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix}$:

$$\begin{bmatrix} -a' & 1 & -b' \end{bmatrix} \cdot \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = 0$$
$$\vec{c'}^T \cdot \vec{p'} = 0$$

However, since we know the transformation that was used to obtain the points $p' = (\frac{x'}{w'}, \frac{y'}{w'})$, we can substitute that back into the line equation for the new points to get a line equation using the original points:

$$\vec{c'}^T \cdot \vec{p'} = 0$$
$$\vec{c'}^T \cdot \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = 0$$
$$\vec{c'}^T \cdot \left(A \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \right) = 0$$
$$\left(\vec{c'}^T \cdot A \right) \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0$$
$$\left(\vec{c'}^T \cdot A \right) \cdot \vec{p} = 0$$

This looks very similar to the matrix notation for the original arbitrary line $y = ax + b$.

We want a non-trivial solution such that this equation is still true but $\vec{p} \neq \vec{0}$. Comparing this with the equation $\vec{c}^T \cdot \vec{p} = 0$ from above, we try to substitue $\vec{c'}^T = \vec{c}^T \cdot \left(A^{-1}\right)$.

The points $p' = (\frac{x'}{w'}, \frac{y'}{w'})$ form a unique line. This means the vector $\vec{c'}$ to make that line is also unique. Finally, the substitution $\vec{c'}^T = \vec{c}^T \cdot \left(A^{-1}\right)$ must as a consequence also be a unqiue solution.

Using the substitution here:

$$\begin{aligned} \vec{c'}^T \cdot A \cdot \vec{p} &= 0 \\ (\vec{c}^T \cdot A^{-1}) \cdot A \cdot \vec{p} &= 0 \\ \vec{c}^T \cdot \left(A^{-1} \cdot A\right) \cdot \vec{p} &= 0 \\ \vec{c}^T \cdot \vec{p} &= 0 \\ \begin{bmatrix} -a & 1 & -b \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} &= 0 \end{aligned}$$

We notice that the resulting line $y' = a'x' + b'w'$ defines the same line originally defined by $y = ax + b$.

This means that for affine transformation matrix A with $ae \neq bd$ and $\vec{c'} = \left(A^{-1}\right)^T \cdot \vec{c}$, the new line $y' = a'x' + b'w'$ is equal to the original line $y = ax + b$ meaning lines map to lines.

Problem 2

Prove that any roto-reflective transformation $R^2 \rightarrow R^2$ defined by 2x2 orthogonal matrix R (s.t. $R^TR = I$) preserves (a) orthogonal lines and (b) distances between points. Your proof should use only linear algebraic equations.

Solution:

For part a:

Take some vector $\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$ and and another vector that we know is orthogonal to it $\begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$ both in R^2.

This means the scalar product between them is 0. i.e.:

$$\begin{aligned} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \cdot \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} &= 0 \\ \left(\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}\right)^T \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} &= 0 \\ \begin{bmatrix} x_1 & y_1 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} &= 0 \end{aligned}$$

The transformation of these vectors are as follows:

$$R \cdot \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} x'_1 \\ y'_1 \end{bmatrix}$$

$$R \cdot \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x'_2 \\ y'_2 \end{bmatrix}$$

Finding the scalar product between these new vectors:

$$\begin{aligned}
\begin{bmatrix} x'_1 \\ y'_1 \end{bmatrix} \cdot \begin{bmatrix} x'_2 \\ y'_2 \end{bmatrix} &= 0 \\
\left(\begin{bmatrix} x'_1 \\ y'_1 \end{bmatrix}\right)^T \begin{bmatrix} x'_2 \\ y'_2 \end{bmatrix} &= 0 \\
\left(R \cdot \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}\right)^T \left(R \cdot \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}\right) &= 0 \\
\left(\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}^T R^T\right) \left(R \cdot \begin{matrix} x_2 \\ y_2 \end{matrix}\right) &= 0 \\
\begin{bmatrix} x_1 & y_1 \end{bmatrix} R^T R \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} &= 0 \\
\begin{bmatrix} x_1 & y_1 \end{bmatrix} (I) \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} &= 0 \\
\begin{bmatrix} x_1 & y_1 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} &= 0
\end{aligned}$$

This shows that even after transformation from a roto-reflective transformation, orthogonal lines are preserved.

For part b)

Let X and Y be points in R^2.

The euclidean distance between them gives the distance squared between them as the sum of the squares of the distances along the two axes, that is:

$$\begin{aligned}
d(X, Y)^2 &= (X_1 - Y_1)^2 + (X_2 - Y_2)^2 \\
&= (X - Y) \cdot (X - Y) \\
&= (X - Y)^T (X - Y)
\end{aligned}$$

In other words, it is the scalar product between the vector formed by subtracting the points and itself.

Applying the transformation R to it:

$$\begin{aligned} d(R(X),R(Y))^2 &= (R(X)-R(Y))\cdot(R(X)-R(Y)) \\ &= R(X-Y)\cdot R(X-Y) \\ &= (R(X-Y))^T R(X-Y) \\ &= ((X-Y)^T R^T)R(X-Y) \\ &= (X-Y)^T(R^T R)(X-Y) \\ &= (X-Y)^T(I)(X-Y) \\ &= (X-Y)^T(X-Y) \end{aligned}$$

Since $d(X,Y)^2$ is equal to $d(R(X),R(Y))^2$ distance between points are preserved even after transformation from a roto-reflective transformation

Problem 3 (least-squares)

Complete implementation of function $estimate$ of class $LeastSquareLine$ below. It should update line parameters a and b correpsonding to line model $y = ax + b$. You can use either SVD of matrix A or inverse of matrix $A^T A$, as mentioned in class. NOTE: several cells below test your code.

In [1]:
```
%matplotlib notebook

import numpy as np
import numpy.linalg as la
import matplotlib
import matplotlib.pyplot as plt
from skimage.measure import ransac
```

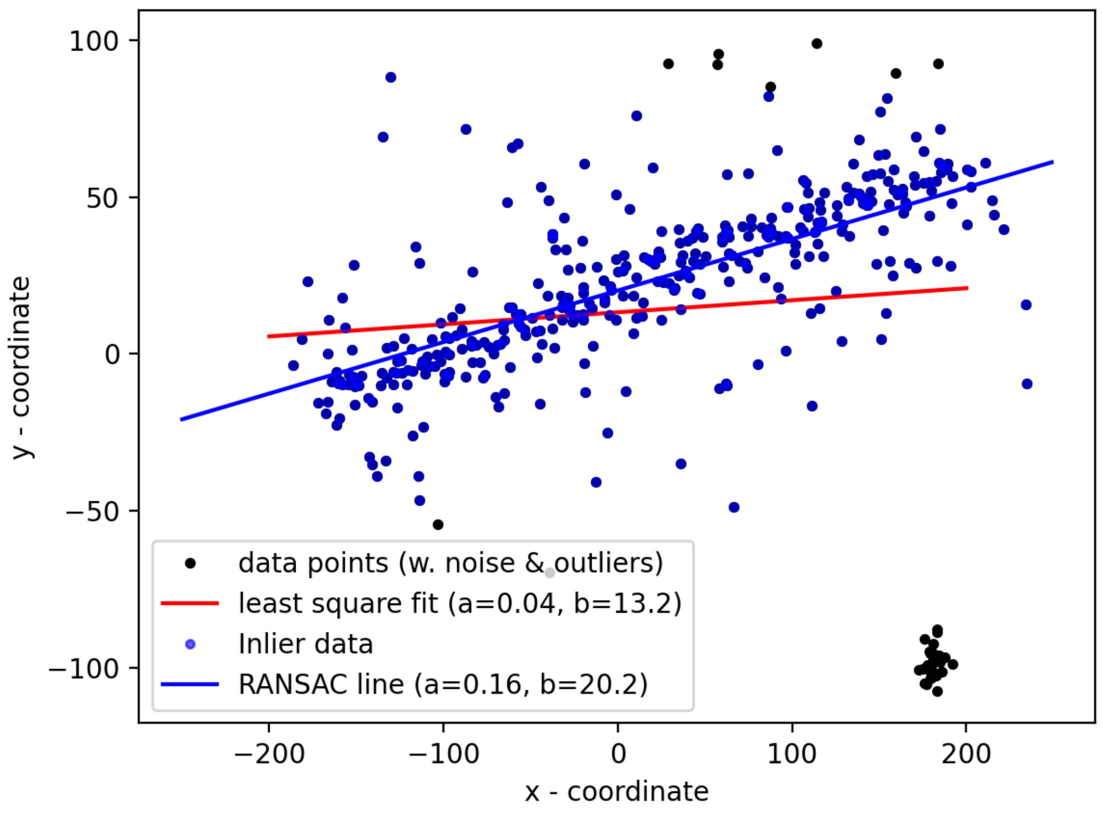
In [2]:
```
class LeastSquareLine:

    def __init__(self):
        self.a = 0.0
        self.b = 0.0

    def estimate(self, points2D):
        B = points2D[:,1]
        A = np.copy(points2D)
        A[:,1] = 1.0

        # Vector B and matrix A are already defined. Change code below
        x = np.linalg.pinv(A.T @ A) @ A.T @ B
#         u, s, vh = np.linalg.svd(A, full_matrices=False)
#         x = vh @ np.linalg.pinv(np.diag(s)) @ u.T @ B

        self.a = x[0]
        self.b = x[1]
        return True

    def predict(self, x): return (self.a * x) + self.b

    def predict_y(self, x): return (self.a * x) + self.b
    def residuals(self, points2D):
        return points2D[:,1] - self.predict(points2D[:,0])

    def line_par(self):
        return self.a, self.b
```

Problem 4 (RANSAC for robust line fitting, single model)

Working code below generates a noisy cloud of points in $\mathcal{R}^2$ from a given line and a group of outliers.

In [3]:

```
np.random.seed(seed=1)

# parameters for "true" line y = a*x + b
a, b = 0.2, 20.0

# x-range of points [x1,x2]
x_start, x_end = -200.0, 200.0

# generate "idealized" line points
x = np.arange(x_start,x_end)
y = a * x + b
data = np.column_stack([x, y])    # staking data points into (Nx2) array

# add gaussian pertubations to generate "realistic" line points
noise = np.random.normal(size=data.shape) # generating Gaussian noise (variance
data += 5 * noise
data[::2] += 10 * noise[::2]  # every second point adds noise with variance 5
data[::4] += 20 * noise[::4] # every fourth point adds noise with variance 20

# add outliers
faulty = np.array(30 * [(180., -100)])  # (30x2) array containing 30 rows [180,-
faulty += 5 * np.random.normal(size=faulty.shape)  # adding Gaussian noise to th
data[:faulty.shape[0]] = faulty   # replacing the first 30 points in data with f


fig, ax = plt.subplots()
ax.plot(data[:,0], data[:,1], '.k', label='data points (w. noise & outliers)')
ax.set_xlabel('x - coordinate')
ax.set_ylabel('y - coordinate')
ax.legend(loc='lower left')
plt.show()
```



Code below uses your implementation of class $LeastSquareLine$ for least-square line fitting for the data above. The estimated line is displayed in the cell above. Use this cell to test your code in Problem 2. Of course, your least-square line will be affected by the outliers.

In [4]:

```
LSline = LeastSquareLine() # uses class implemented in Problem 2
print (LSline.estimate(data))
a_ls, b_ls = LSline.line_par()

# visualizing estimated line
ends = np.array([x_start,x_end])
ax.plot(
    ends,
    LSline.predict(ends),
    '-r',
    label='least square fit (a={:4.2f}, b={:4.1f})'.format(a_ls,b_ls)
)
ax.legend(loc='lower left')
plt.show()
```

```
True
```

(part a) Assume that a set of $N = 100$ points in $2D$ includes $N_i = 20$ inliers for one line and $N_o = 80$ outliers. What is the least number of times one should sample a random pair of points from the set to get probability $p \geq 0.95$ that in at least one of the sampled

pairs both points are inliers? Derive a general formula and compute a numerical answer for the specified numbers.

Solution:

Let I be "Inliers" and O be outliers and n be least number of trials one should randomly sample a subset of size 2 such that both samples are inliers.

$$\begin{aligned} P(\text{at least 1 trial with 2 I in n trials}) &= 1 - P(\text{never get 1 trial with 2 I in n trials}) \\ &= 1 - (P(\text{do NOT get 2I}))^n \\ &= 1 - (1 - P(\text{DO get 2I}))^n \\ &= 1 - \left(1 - \frac{\binom{I}{2}}{\binom{I}{2} + \binom{O}{2} + \binom{O}{1}\cdot\binom{I}{1}}\right)^n \\ &>= 0.95 \end{aligned}$$

Simplifying leads to the following:

$$\begin{aligned} 1 - \left(1 - \frac{\binom{I}{2}}{\binom{I}{2} + \binom{O}{2} + \binom{O}{1}\cdot\binom{I}{1}}\right)^n &>= 0.95 \\ \left(1 - \frac{\binom{I}{2}}{\binom{I}{2} + \binom{O}{2} + \binom{O}{1}\cdot\binom{I}{1}}\right)^n &<= 0.05 \\ \left(1 - \frac{\frac{I(I-1)}{2}}{\frac{I(I-1)}{2} + \frac{O(O-1)}{2} + IO}\right)^n &<= 0.05 \\ \left(1 - \frac{I(I-1)}{I(I-1) + O(O-1) + 2IO}\right)^n &<= 0.05 \end{aligned}$$

Let $a = P\left(\text{DO get 2I}\right) = \frac{I(I-1)}{I(I-1)+O(O-1)+2IO}$.

The equation becomes:

$$\begin{aligned} \left(1 - \frac{I(I-1)}{I(I-1) + O(O-1) + 2IO}\right)^n &<= 0.05 \\ (1-a)^n &<= 0.05 \\ log_{(1-a)}(1-a)^n &>= log_{(1-a)}0.05 \\ n &>= log_{(1-a)}0.05 \\ n &>= \frac{\log 0.05}{\log(1-a)} \\ n &>= \frac{\log 0.05}{\log\left(1 - \frac{I(I-1)}{I(I-1)+O(O-1)+2IO}\right)} \end{aligned}$$

Solving for $I = 20$ and $O = 80$ as in the question:

$$n >= \frac{\log 0.05}{\log\left(1 - \frac{I(I-1)}{I(I-1)+O(O-1)+2IO}\right)}$$

$$n >= \frac{\log 0.05}{\log\left(1 - \frac{(20)((20)-1)}{(20)((20)-1)+(80)((80)-1)+2(20)(80)}\right)}$$

$$n >= \frac{\log 0.05}{\log\left(1 - \frac{380}{9{,}520}\right)}$$

$$n >\approx 73.54$$

Therefore 74 is the minimum number of samples n that we must make to have a probability $p >= 0.95$ that at least one of the sampled pairs both have inliers.

(part b) Using the knowledge of the number of inliers/outliers in the example at the beginning of Problem 3, estimate the minimum number of sampled pairs needed to get RANSAC to "succeed" (to get at least one pair of inliers) with $p \geq 0.95$. Use your formula in part (a). Show your numbers in the cell below. Then, use your estimate as a value of parameter max_trials inside function $ransac$ in the code cell below and test it. You should also change $residual\_threshold$ according to the noise level for inliers in the example. NOTE: the result is displayed in the same figure at the beginning of Problem 3.

Your estimates:

There are `np.arange(-200, 200) = 400` originally idealized points. Half of them have high variance leaving only 200 good points. Of those 200 good inlier points 15 of them were replaced when 30 outlier points were added.

Using the formula from part a):

Solving for $I = 185$ and $O = 30$ as in the question:

$$n >= \frac{\log 0.05}{\log\left(1 - \frac{(185)((185)-1)}{(185)((185)-1)+(30)((30)-1)+2(185)(30)}\right)}$$

$$n >= \frac{\log 0.05}{\log\left(1 - \frac{34{,}040}{46{,}010}\right)}$$

$$n >\approx 2.22$$

Therefore 3 is the minimum number of samples n that we must make to have a probability $p >= 0.95$ that at least one of the sampled pairs both have inliers.

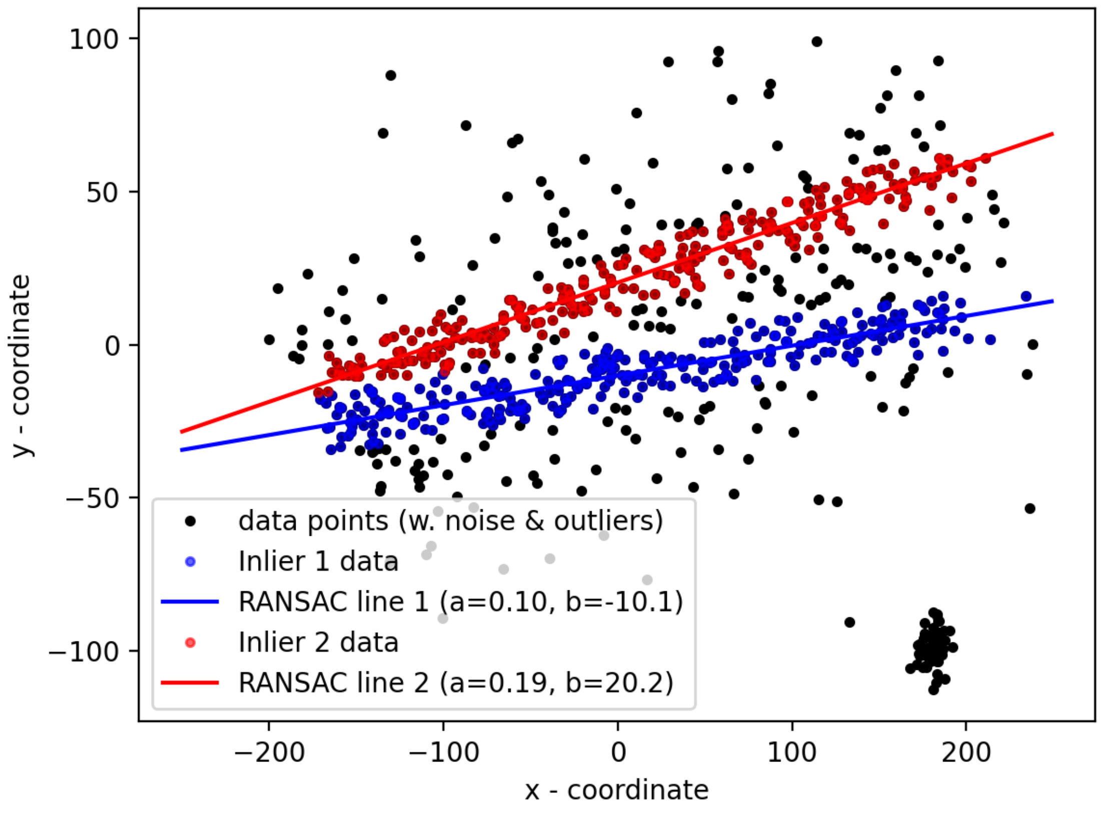
In [5]:
```python
# Including 2 standard deviations of noise should include 95% of points under th
thresh = 2*np.std(5 * noise + 10 * noise + 20 * noise)

# robustly fit line using RANSAC algorithm
model_robust, inliers = ransac(
    data,
    LeastSquareLine,
    min_samples=2,
    residual_threshold=thresh,
    max_trials=3
)
a_rs, b_rs = model_robust.line_par()

# generate coordinates of estimated models
line_x = np.arange(-250, 250)
line_y_robust = model_robust.predict_y(line_x)

#fig, ax = plt.subplots()
ax.plot(
    data[inliers, 0],
    data[inliers, 1],
    '.b',
    alpha=0.6,
    label='Inlier data'
)
ax.plot(
    line_x,
    line_y_robust,
    '-b',
    label='RANSAC line (a={:4.2f}, b={:4.1f})'.format(a_rs,b_rs)
)
ax.legend(loc='lower left')
plt.show()
```

Problem 5 (sequential RANSAC for robust multi-line fitting)

Generating noisy data with outliers

In [6]:

```
# parameters for "true" lines y = a*x + b
a2, b2 = 0.1, -10.0

# generate "idealized" line points
y2 = a2 * x + b2
data2 = np.column_stack([x, y2])    # staking data points into (Nx2) array

# add gaussian pertubations to generate "realistic" line data
noise = np.random.normal(size=data.shape) # generating Gaussian noise (variance
data2+= 5 * noise
data2[::2] += 10 * noise[::2]  # every second point adds noise with variance 5
data2[::4] += 20 * noise[::4] # every fourth point adds noise with variance 20

# add outliers
faulty = np.array(30 * [(180., -100)])  # (30x2) array containing 30 rows [180,-
faulty += 5 * np.random.normal(size=faulty.shape)  # adding Gaussian noise to th
data2[:faulty.shape[0]] = faulty   # replacing the first 30 points in data2 with

data = np.concatenate((data,data2)) # combining with previous data

fig, ax = plt.subplots()
ax.plot(
    data[:,0],
    data[:,1],
    '.k',
    label='data points (w. noise & outliers)'
)
ax.set_xlabel('x - coordinate')
ax.set_ylabel('y - coordinate')
ax.legend(loc='lower left')
plt.show()
```



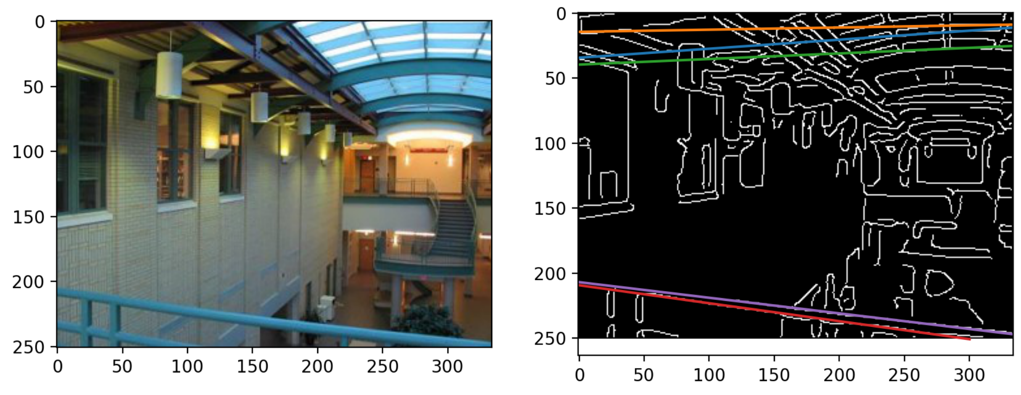
Write code below using sequential RANSAC to detect two lines in the data above. Your lines should be displayed in the figure above.

In [7]:
```
# Because the lines have so much overlap, we need to emperically
# determine what an appropriate `residual_threshold` value would be

# robustly fit line using RANSAC algorithm
model_robust_1, inliers_1 = ransac(
    data,
    LeastSquareLine,
    min_samples=2,
    residual_threshold=thresh/7,
    # We take many more sample here because the 2 lines
    # overlap and become outliers for each other.
    max_trials=200
)
a_rs_1, b_rs_1 = model_robust_1.line_par()

data_minus_1 = np.delete(data, inliers_1, axis=0)

model_robust_2, inliers_2 = ransac(
    data_minus_1,
    LeastSquareLine,
    min_samples=2,
    residual_threshold=thresh/7,
    max_trials=200
)
a_rs_2, b_rs_2 = model_robust_2.line_par()

# generate coordinates of estimated models
line_x = np.arange(-250, 250)
line_y_robust_1 = model_robust_1.predict_y(line_x)
line_y_robust_2 = model_robust_2.predict_y(line_x)

#fig, ax = plt.subplots()
ax.plot(
    data[inliers_1, 0],
    data[inliers_1, 1],
    ".b",
    alpha=0.6,
    label="Inlier 1 data",
)
ax.plot(
    line_x,
    line_y_robust_1,
    "-b",
    label="RANSAC line 1 (a={:4.2f}, b={:4.1f})".format(
        a_rs_1, b_rs_1
    ),
)
ax.plot(
    data_minus_1[inliers_2, 0],
    data_minus_1[inliers_2, 1],
    ".r",
    alpha=0.6,
    label="Inlier 2 data",
)
ax.plot(
    line_x,
    line_y_robust_2,
    "-r",
    label="RANSAC line 2 (a={:4.2f}, b={:4.1f})".format(
        a_rs_2, b_rs_2
    ),
)
ax.legend(loc="lower left")
plt.show()
```

Problem 6 (multi-line fitting for Canny edges)

In [8]:

```
import matplotlib.image as image
from skimage import feature
from skimage.color import rgb2gray

im = image.imread("images/CMU_left.jpg")
imgray = rgb2gray(im)
can = feature.canny(imgray, 2.0)

plt.figure(3,figsize = (10, 4))
plt.subplot(121)
plt.imshow(im)
ax = plt.subplot(122)
plt.imshow(can,cmap="gray")
plt.show()
```



use sequestial-RANSAC to find K lines

In [9]:
```
K = 5
# NOTE 1: write your code using a function that takes K as a parameter.
# NOTE 2: Present visual results for some value of K
# NOTE 3: Your code should visually show detected lines in a figure
#         over the image (either the original one or over the Canny edge mask)
# NOTE 4: You may need to play with parameters of function ransac
#         (e.g. threshold and number of sampled models "max_trials")
#         Also, you can introduce one extra parameter for the minimum number of
#         for accepting ransac-detected lines.

# NOTE: "can" in the cell above is a binary mask with True and False values, e.g
print (can[20,30])
print (can[146,78])
```

```
False
True
```

In [10]:
```
def draw_k_lines(K, data):
    data_coords = np.roll(
        np.argwhere(data), 1
    )
    line_x = np.arange(0, data.shape[1])
    for i in range(K):
        inliers_k = np.array([0])
        while (
            inliers_k is None
            or np.sum(inliers_k)
            < -4 * K + 160
        ):
            (
                model_robust_k,
                inliers_k,
            ) = ransac(
                data_coords,
                LeastSquareLine,
                min_samples=2,
                residual_threshold=1,
                max_trials=200,
            )
            print(
                "Found ",
                0
                if inliers_k is None
                else np.sum(inliers_k),
                " inliers.",
                end="\r",
            )
        print("")
        old_data_coords_inliers = (
            data_coords.shape[0]
        )
        data_coords = np.delete(
            data_coords,
            inliers_k,
            axis=0,
        )
        print(
            "Removed ",
            old_data_coords_inliers
            - data_coords.shape[0],
            " inliers.",
        )
        (
            a_rs_k,
            b_rs_k,
        ) = model_robust_k.line_par()
        print(
            f"Formed this line #{i + 1}: y = {a_rs_k}x + {b_rs_k}"
        )

        # generate coordinates of estimated models
        line_y_robust_k = (
            model_robust_k.predict_y(
                line_x
            )
        )
        in_range_indices = (
            line_y_robust_k
            < data.shape[0]
        ) * (line_y_robust_k >= 0)
        x_vals = line_x[
            in_range_indices
        ]
        y_vals = line_y_robust_k[
            in_range_indices
        ]
        ax.plot(
            x_vals,
            y_vals,
            label="RANSAC line {} (a={:4.2f}, b={:4.1f})".format(
                i, a_rs_k, b_rs_k
            ),
        )
    plt.show()
    print(f"Done finding {K} lines.")


draw_k_lines(K, can)
```

```
Found  140  inliers.
Removed  140  inliers.
Formed this line #1: y = -0.06940144639582896x + 34.36020966244973
Found  149  inliers.
Removed  149  inliers.
Formed this line #2: y = -0.016669904179093486x + 14.444364551601591
Found  140  inliers.
Removed  140  inliers.
Formed this line #3: y = -0.04299706676129955x + 39.64537807963424
Found  185  inliers.
Removed  185  inliers.
Formed this line #4: y = 0.13830762685844367x + 209.38461360770518
Found  140  inliers.
Removed  140  inliers.
Formed this line #5: y = 0.12011096210335528x + 206.99969175666348
Done finding 5 lines.
```