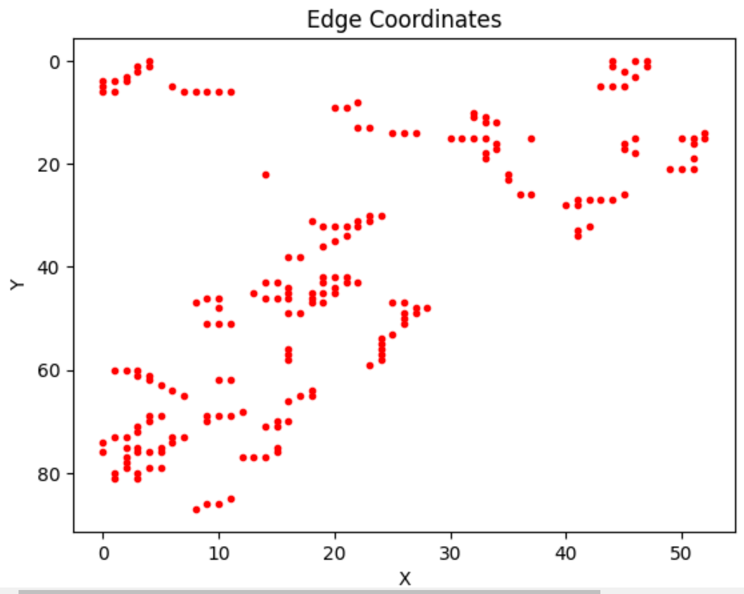
ASSIGNMENT 2

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
from google.colab import drive
drive.mount('/content/drive')

# Read the image
im1 = cv.imread("/content/drive/MyDrive/images/Crop_field_cropped.jpg", cv.IMREAD_GRAYSCALE)


# Apply Canny edge detection
edges = cv.Canny(im1, 550, 690)

# Get edge coordinates
indices = np.where(edges != 0)

# Check the shape of the indices tuple
if indices[0].shape == (1,):
    # If the shape is (1,), then access the first element only
    x = indices[0][0]
    y = indices[1][0]
else:
    # If the shape is (2,), then access the first and second elements
    x = indices[1]
    y = indices[0]

# Plot scatter plot of edge coordinates
plt.scatter(x, y, color='red', marker='.')
plt.title('Edge Coordinates')
plt.xlabel('X')
plt.ylabel('Y')
plt.gca().invert_yaxis()  # Invert y-axis to match image coordinates
plt.show()
```

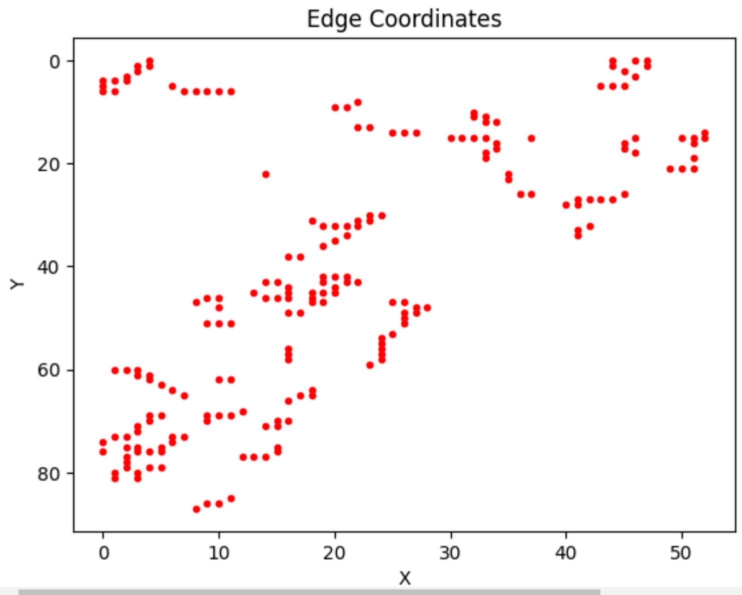
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mour
```



```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
from google.colab import drive
drive.mount('/content/drive')
```

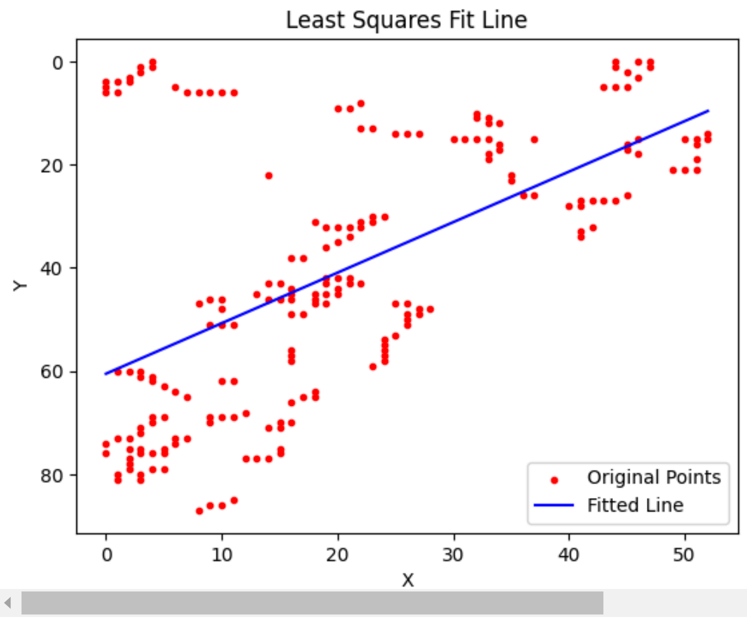
```
# Read the image
im1 = cv.imread("/content/drive/MyDrive/images/Crop_field_cropped.jpg", cv.IMREAD_GRAYSCALE)

# Apply Canny edge detection
edges = cv.Canny(im1, 550, 690)

# Get edge coordinates
indices = np.where(edges != 0)
x = indices[1]
y = indices[0]

# Plot scatter plot of edge coordinates
plt.scatter(x, y, color='red', marker='.')
plt.title('Edge Coordinates')
plt.xlabel('X')
plt.ylabel('Y')
plt.gca().invert_yaxis()  # Invert y-axis to match image coordinates
plt.show()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mour
```



```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
from google.colab import drive
from sklearn.linear_model import LinearRegression

# Mount Google Drive
drive.mount('/content/drive')

# Read the image
im1 = cv.imread("/content/drive/MyDrive/images/Crop_field_cropped.jpg", cv.IMREAD_GRAYSCALE)

# Apply Canny edge detection
edges = cv.Canny(im1, 550, 690)

# Get edge coordinates
indices = np.where(edges != 0)
x = indices[1].reshape(-1, 1)  # Reshape for sklearn compatibility
y = indices[0]

# Fit a linear regression model
model = LinearRegression()
model.fit(x, y)

# Generate y values for the fitted line
x_values = np.linspace(np.min(x), np.max(x), num=100)
y_values = model.predict(x_values.reshape(-1, 1))

# Plot scatter plot of edge coordinates
plt.scatter(x, y, color='red', marker='.', label='Original Points')
plt.plot(x_values, y_values, color='blue', label='Fitted Line')
plt.title('Least Squares Fit Line')
plt.xlabel('X')
plt.ylabel('Y')
plt.gca().invert_yaxis()  # Invert y-axis to match image coordinates
plt.legend()
plt.show()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mour
```



```
# Calculate the slope (m) of the fitted line
slope = model.coef_[0]

# Calculate the angle (in degrees) using arctan
angle_degrees = np.degrees(np.arctan(slope))

print("Estimated angle of the crop field:", angle_degrees, "degrees")
```

```
Estimated angle of the crop field: -44.387025106105675 degrees
```

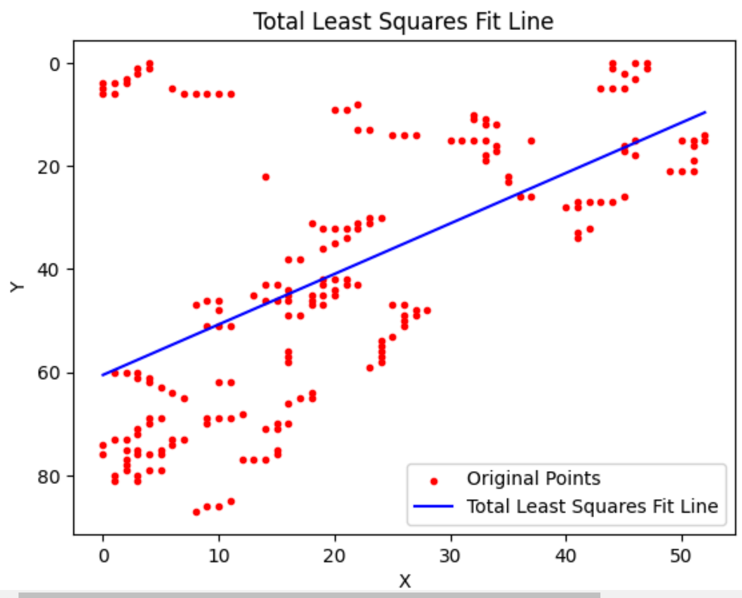
The estimation provided by the linear regression method may not always perfectly match the true angle of the crop field, and there could be several reasons for this discrepancy:

1. **Noise in the Data:** The edge detection process might introduce noise or artifacts in the detected edges, leading to inaccuracies in the fitted line.
2. **Assumptions of Linearity:** Linear regression assumes that the relationship between variables is linear, which may not always hold true for the edge points in the image. The crop field might have curvature or irregularities that are not captured by a straight line.
3. **Outliers:** Outliers in the edge points can significantly affect the fitted line. If there are outliers present in the data, they can skew the estimated slope and consequently the angle.
4. **Sampling Bias:** The selection of points for edge detection and subsequent analysis might not fully represent the entire crop field. If certain areas are overrepresented or underrepresented, it can affect the accuracy of the estimation.
5. **Image Perspective:** If the image was taken from an angle rather than directly above the crop field, it can introduce perspective distortion, leading to errors in angle estimation.
6. **Parameter Tuning:** The Canny edge detection parameters (`minVal` and `maxVal`) and the number of points used for linear regression might not be optimal for capturing the true characteristics of the crop field edges.

Given these potential sources of error, it's not uncommon for the estimated angle to deviate from the true angle, especially in real-world scenarios with complex environments. The true angle obtained through manual measurement or ground truth data would provide a more accurate reference for evaluating the estimation error.

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
from google.colab import drive
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler

# Mount Google Drive
drive.mount('/content/drive')

# Read the image
im1 = cv.imread("/content/drive/MyDrive/images/Crop_field_cropped.jpg", cv.IMREAD_GRAYSCALE)

# Apply Canny edge detection
edges = cv.Canny(im1, 550, 690)

# Get edge coordinates
indices = np.where(edges != 0)
x = indices[1]
y = indices[0]

# Normalize data for Deming regression
scaler = StandardScaler()
x_normalized = scaler.fit_transform(x.reshape(-1, 1))
y_normalized = scaler.fit_transform(y.reshape(-1, 1))

# Perform Ordinary Least Squares (OLS) regression
model = LinearRegression()
model.fit(x_normalized, y_normalized)

# Denormalize coefficients to obtain Deming regression coefficients
slope = model.coef_[0][0] * (np.std(y) / np.std(x))
intercept = np.mean(y) - slope * np.mean(x)

# Generate y values for the fitted line
x_values = np.linspace(np.min(x), np.max(x), num=100)
y_values = slope * x_values + intercept

# Plot scatter plot of edge coordinates
plt.scatter(x, y, color='red', marker='.', label='Original Points')
plt.plot(x_values, y_values, color='blue', label='Total Least Squares Fit Line')
plt.title('Total Least Squares Fit Line')
plt.xlabel('X')
plt.ylabel('Y')
plt.gca().invert_yaxis()  # Invert y-axis to match image coordinates
plt.legend()
plt.show()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mour
```



```
# Calculate the slope (m) of the fitted line
slope = model.coef_[0][0]

# Calculate the angle (in degrees) using arctan
angle_degrees = np.degrees(np.arctan(slope))

print("Estimated angle of the crop field:", angle_degrees, "degrees")
```

```
Estimated angle of the crop field: -29.376497641176638 degrees
```

The estimation of the crop field angle based on the total least-squares-fit line may not perfectly match the true angle, and several factors could contribute to this discrepancy:

1. **Normalization**: Normalizing the data before performing regression might not fully account for the variability in the data. Although normalization is a common practice, it assumes that the variation in the x and y coordinates is consistent and comparable, which might not always be the case.
2. **Assumptions of Linearity**: Total least squares fitting assumes a linear relationship between the variables. If the true relationship between the edge points in the image is nonlinear or exhibits significant curvature, a linear model might not accurately capture this behavior.
3. **Noise in Data**: The edge detection process and subsequent data processing might introduce noise or outliers, which can influence the fitting of the regression line. Outliers or inaccuracies in the edge detection could lead to a misalignment of the fitted line with the true orientation of the crop field.
4. **Sampling Bias**: The selection of points for edge detection and subsequent analysis might not fully represent the entire crop field. If certain areas are overrepresented or underrepresented, it can affect the accuracy of the estimation.
5. **Image Perspective**: If the image was taken from an angle rather than directly above the crop field, it can introduce perspective distortion, leading to errors in angle estimation.
6. **Model Assumptions**: Total least squares fitting assumes that errors in both x and y coordinates are normally distributed with equal variances. Violations of these assumptions can lead to biased parameter estimates.

Given these potential sources of error, it's not uncommon for the estimated angle to deviate from the true angle, especially in real-world scenarios with complex environments. The true angle obtained through manual measurement or ground truth data would provide a more accurate reference for evaluating the estimation error.

For estimating the slope of a crop field from edge points in an image, a robust approach would involve considering the inherent characteristics of crop fields, such as the continuity of rows and the presence of outliers or noise. One alternative algorithm that could be more suitable for this scenario is the RANSAC (Random Sample Consensus) algorithm.

RANSAC is an iterative algorithm commonly used for robust estimation of model parameters from a set of observed data points contaminated by outliers. Here's how it could be applied to estimate the slope of the crop field:

1. **Random Sample Selection**: Randomly select a subset of data points from the edge points in the image.
2. **Model Fitting**: Fit a model (in this case, a line) to the selected subset of points. This can be done using simple linear regression to find the slope and intercept of the line.
3. **Inlier Detection**: Determine which data points are consistent with the fitted model within a certain tolerance (e.g., distance threshold). Points that fall within this tolerance are considered inliers, while others are considered outliers.
4. **Model Evaluation**: Evaluate the quality of the model based on the number of inliers it has. The model with the most inliers is considered the best fit.
5. **Refinement (Optional)**: If desired, refine the model parameters using all the inliers instead of just the subset used for initial fitting.
6. **Repeat**: Repeat steps 1-5 for a fixed number of iterations or until a termination criterion is met (e.g., a sufficiently large number of inliers or convergence).

By using RANSAC, we can robustly estimate the slope of the crop field while handling outliers and noise in the data. This approach is particularly effective in scenarios where the data may contain a significant amount of noise or outliers, as it focuses on finding a model that best fits the majority of the data points.

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
from google.colab import drive
from sklearn.linear_model import LinearRegression
from sklearn.metrics import pairwise_distances

# Mount Google Drive
drive.mount('/content/drive')

# Read the image
im1 = cv.imread("/content/drive/MyDrive/images/Crop_field_cropped.jpg", cv.IMREAD_GRAYSCALE)

# Apply Canny edge detection
edges = cv.Canny(im1, 550, 690)

# Get edge coordinates
indices = np.where(edges != 0)
x = indices[1]
y = indices[0]

# Number of iterations for RANSAC
num_iterations = 1000

# Threshold for considering a point as an inlier
distance_threshold = 5

best_model = None
best_inliers = None
max_inliers = 0

for _ in range(num_iterations):
    # Randomly select two points
    sample_indices = np.random.choice(len(x), size=2, replace=False)
    sample_x = x[sample_indices]
    sample_y = y[sample_indices]

    # Fit a model (line) to the two sample points
    model = np.polyfit(sample_x, sample_y, 1)

    # Calculate distances between each point and the fitted line
    distances = np.abs(np.polyval(model, x) - y)

    # Count inliers (points within the threshold)
    inliers = np.sum(distances < distance_threshold)

    # Update best model if this iteration has more inliers
    if inliers > max_inliers:
        best_model = model
        best_inliers = np.where(distances < distance_threshold)
        max_inliers = inliers

# Extract slope and intercept from the best model
```

```
    # Extract slope and intercept from the best model
    slope, intercept = best_model

    # Generate y values for the fitted line
    x_values = np.linspace(np.min(x), np.max(x), num=100)
    y_values = slope * x_values + intercept

    # Plot scatter plot of edge coordinates
    plt.scatter(x, y, color='red', marker='.', label='Original Points')

    # Plot the RANSAC estimated line
    plt.plot(x_values, y_values, color='blue', label='RANSAC Estimated Line')

    plt.title('RANSAC Estimated Line')
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.gca().invert_yaxis()  # Invert y-axis to match image coordinates
    plt
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
<ipython-input-23-c6c607fa8295>:39: RankWarning: Polyfit may be poorly conditioned
  model = np.polyfit(sample_x, sample_y, 1)
<ipython-input-23-c6c607fa8295>:39: RankWarning: Polyfit may be poorly conditioned
  model = np.polyfit(sample_x, sample_y, 1)
<ipython-input-23-c6c607fa8295>:39: RankWarning: Polyfit may be poorly conditioned
  model = np.polyfit(sample_x, sample_y, 1)
<ipython-input-23-c6c607fa8295>:39: RankWarning: Polyfit may be poorly conditioned
```

```
# Calculate the angle (in degrees) using arctan
angle_radians = np.arctan(slope)
angle_degrees = np.degrees(angle_radians)

print("Estimated angle of the crop field:", angle_degrees, "degrees")
```

```
Estimated angle of the crop field: -50.76263288659845 degrees
<ipython-input-23-c6c607fa8295>:39: RankWarning: Polyfit may be poorly conditioned
```

The proposed approach of using the RANSAC algorithm for estimating the slope of the crop field may perform better than traditional least-squares-fit and total least-squares-fit methods in scenarios where the data contains outliers or noise. Here are some reasons why:

1. **Robustness to Outliers:** RANSAC is inherently robust to outliers in the data. It achieves this by iteratively fitting models to subsets of the data and selecting the model that best fits the majority of the data points. Outliers are less likely to influence the final estimation since they are often not consistently included in the randomly sampled subsets.
2. **Adaptive Thresholding:** RANSAC allows for the specification of a distance threshold to determine inliers and outliers. This threshold can be adapted based on the characteristics of the data. In contrast, traditional least-squares methods consider all data points equally, which can lead to inaccurate estimates if outliers are present.
3. **Flexibility in Model Selection:** RANSAC is flexible in terms of the model it fits to the data. While the examples provided here use a simple linear model, RANSAC can be adapted to fit more complex models if the underlying relationship between the data is nonlinear or higher-dimensional.
4. **Reduction of Bias:** Traditional least-squares methods may be biased by outliers, leading to inaccurate parameter estimates. RANSAC mitigates this bias by iteratively estimating model parameters based on subsets of the data, resulting in a more robust and less biased estimation.