Guide to calculating the coordinates of each tree

To calculate the coordinates of each tree, we will use the center coordinates you provide and distribute the trees according to the distance between rows and between trees, assuming they are rectangular for ease of calculation.

Data used

- **Center Coordinates** Zone 48P, X = 267665, Y = 1587689
- **Area** 38 rai 1 ngan 77 square wa (converted to square meters: 38.4425 rai * 1600 sq m./rai = 61500 sq m.)
- **Number of trees** : 2,999
- **Row spacing** 8 meters
- **Distance between trees** 3 meters

Calculation steps

1. **Find the approximate size of the plot.** From the number of trees and the planting distance, we can estimate the size of the plot.
    - Approximate number of rows = sqrt(number of trees × distance between trees × distance between rows) = sqrt(2999×38) ≈ sqrt(7997.33) ≈ 89.43 rows
    - Approximate number of trees in each row = sqrt(number of trees × distance between rows) = sqrt(2999×83) ≈ sqrt(1124.625) ≈ 33.53 trees/row
    - **Width of the plot (according to the distance between trees):** 33.53 trees × 3 m ./ tree = 100.59 m.
    - (**according to row spacing):** 89.43 rows ×8 m./row =715.44 m .

Note: The values obtained may not correspond exactly to the area due to rounding and the assumption of a perfect rectangle, but they should be used as a guideline for determining the tree coordinates.

2. **Set starting point** We will define the center coordinate (X=267665, Y=1587689) as the midpoint of the plot, so the starting point of the first row (southwest corner) can be found by:
    - **X_start:** 267665−(width /2) = 267665−(100.59/2) = 267665−50.295 = 267614.705
    - **Y_start:** 1587689−(length /2) = 1587689−(715.44/2) = 1587689−357.72 = 1587331.28

3. **Calculate the coordinates of each tree** From the starting point, we will increase the X and Y values according to the distance between plants and between rows. give i is the row sequence (starting from 0) and j is the first row sequence (starting from 0).
    - **X coordinate of tree (i, j):** Xstart + (j× distance between plants)
    - **Y coordinate of tree (i, j):** Ystart + i × row spacing)

Example of some coordinate calculations (first 10 trees)

To illustrate, I'll show an example of the coordinates of the first 10 trees in the first row, with the X and Y values rounded to two decimal places.

Tree ID	Row Tree (Col)	coordinates (UTM)	coordinate (UTM)
1	1	267614.71	1587331.28
2	1	267617.71	1587331.28
3	1	267620.71	1587331.28
4	1	267623.71	1587331.28
5	1	267626.71	1587331.28
6	1	267629.71	1587331.28
7	1	267632.71	1587331.28
8	1	267635.71	1587331.28
9	1	267638.71	1587331.28
10	1	267641.71	1587331.28

Creating a total of 2,999 coordinate data points

Calculating all 2,999 coordinates by hand can be time-consuming and error-prone. Your best bet is to use a program or spreadsheet (e.g., Microsoft Excel, Google Sheets) or write some simple code (e.g., Python) to generate a list of all coordinates.

Ideas for working in Excel/Google Sheets

1. Create columns "Tree ID", "Row", "Column", " X_coord ", " Y_coord ".
2. Configure Xstart and Ystart In separate cells
3. row and column sequences (e.g., use the ROUNDUP(ROW ()/ number of rows) function for rows and MOD(ROW()-1, number of rows)+ 1 for columns).
4. Use the formula calculated above to find Xcoord and Ycoord Based on Xstart , Ystart , and Row, Column values

Practical application in rubber plantation management program

Once you have the coordinates of each tree, you can use this information in your rubber plantation management program as follows:

1. **RAOT GIS / Other GIS programs**

- **Import data** Most GIS programs can directly import coordinate data (X, Y) that is in tabular format (such as CSV) .
- **Show results on the map** Once imported, the coordinates of each tree will appear on a map of your rubber plantation.
- **Bind additional information** You can attach additional information to each point, such as rubber variety, tapping age, yield, or fertilizer application history.

- **Spatial data analysis** Use GIS analysis functions to view product distribution patterns or problem areas to plan effective management.

2. **Real-world location identification**

- **Survey** If you have a GPS device , these coordinates can actually be used as targets to survey and locate individual trees in your garden.

- **Rubber tapping/maintenance** This coordinate data can be used to systematically plan rubber tapping routes or maintain individual trees.

Considerations

- **GPS accuracy** the accuracy of GPS measured coordinates may vary depending on the device and environment.

- **Shape transformation** This calculation assumes a rectangular plot. If the plot has a more complex shape, slight adjustments to the coordinates or the use of actual surveying may be required.

- **Dead/replanted trees** Over time, trees may die or be replanted. The coordinate data in the system must be updated to ensure accuracy.

Python V.3.13 program

(Example Data: Calculate tree coordinates TreeProgram.py and example Python code to create a CSV file with coordinates of all 2,999 trees.

```
    # Let's define the number of columns (trees per row) and rows
    # We'll aim for a number of columns that makes sense for the given spacing.
    # For simplicity and to ensure we generate enough trees, let's make the number of columns
    # slightly more than the square root, and calculate rows from that.

    # A more robust way to determine num_rows and num_cols to ensure all trees fit
    # and the aspect ratio is somewhat balanced with the spacing.
    # We want (num_cols * tree_spacing) / (num_rows * row_spacing) to be close to 1
    # And num_cols * num_rows >= total_trees

    # Let's try to make the number of columns such that the total width is reasonable.
    # We will use a fixed number of columns that results in a manageable plot width.
    # For 2999 trees, if we have, say, 50 trees per row, that's about 60 rows.
    # 50 trees * 3m/tree = 150m width
    # 60 rows * 8m/row = 480m length
    # This seems reasonable. Let's adjust to ensure we cover 2999 trees.

    # Let's set a target number of columns, e.g., 55 trees per row.
    # This will give a width of (55-1)*3 = 162m
    num_cols = 55
    num_rows = math.ceil(total_trees / num_cols) # Calculate rows needed to fit all trees

    print(f"Estimated number of columns (trees per row): {num_cols}")
    print(f"Calculated number of rows: {num_rows}")
    print(f"Total grid spots generated: {num_cols * num_rows}")

    # Calculate the total dimensions of the rectangular plot
    # The total length/width is based on the distance between the *first* and *last* tree/row.
    # So, it's (number_of_items - 1) * spacing
    plot_width = (num_cols - 1) * tree_spacing # Total width along X-axis
    plot_length = (num_rows - 1) * row_spacing # Total length along Y-axis

    # Calculate the starting point (bottom-left corner) of the plot
    # Assuming the center_x, center_y is the true geometric center.
    start_x = center_x - (plot_width / 2)
    start_y = center_y - (plot_length / 2)

    tree_data = []
    tree_id_counter = 1


    # Iterate through rows and columns to generate coordinates
    for row_idx in range(num_rows):
        for col_idx in range(num_cols):
            if tree_id_counter > total_trees:
                break # Stop if we have generated enough trees

            # Calculate X and Y coordinates for the current tree
            # X increases as we move right (along columns)
            # Y increases as we move up (along rows)
            current_x = start_x + (col_idx * tree_spacing)
            current_y = start_y + (row_idx * row_spacing)

            tree_data.append({
                "Tree ID": tree_id_counter,
                "Row": row_idx + 1, # Start row number from 1
                "Column": col_idx + 1, # Start column number from 1
                "UTM_X": round(current_x, 2), # Round to 2 decimal places for readability
                "UTM_Y": round(current_y, 2)
            })
            tree_id_counter += 1
        if tree_id_counter > total_trees:
            break

    return tree_data

def save_to_csv(data: list[dict], filename: str):
    """
    Saves the generated tree data to a CSV file.

    Args:
        data (list[dict]): The list of tree data dictionaries.
        filename (str): The name of the CSV file to save.
    """
    if not data:
        print("No data to save.")
        return

    # Get header from the keys of the first dictionary
    fieldnames = data[0].keys()
```

```
    try:
        with open(filename, 'w', newline='', encoding='utf-8') as csvfile:
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader() # Write the column headers
            writer.writerows(data) # Write all the tree data rows
        print(f"Successfully saved {len(data)} tree coordinates to '{filename}'")
    except IOError as e:
        print(f"Error saving to CSV file: {e}")

# --- Configuration ---
# พิกัดศูนย์กลางของสวนยางพารา (UTM Zone 48P)
center_utm_x = 264830.0
center_utm_y = 1582035.0

# จำนวนต้นไม้ทั้งหมด
total_trees_in_plot = 1209

# ระยะห่างระหว่างแถวและระหว่างต้น (เมตร)
row_spacing_meters = 8.0
tree_spacing_meters = 3.0

# ชื่อไฟล์ CSV ที่จะบันทึก
output_csv_filename = "rubber_tree_coordinates.csv"

# --- Generate and Save ---
print("Generating rubber tree coordinates...")
tree_coordinates = generate_rubber_tree_coordinates(
    center_utm_x,
    center_utm_y,
    total_trees_in_plot,
    row_spacing_meters,
    tree_spacing_meters
)

if tree_coordinates:
    save_to_csv(tree_coordinates, output_csv_filename)
else:
    print("Failed to generate tree coordinates.")
```

Python code sample to create a CSV file

```
import csv
import math


def generate_rubber_tree_coordinates(
    center_x: float,
    center_y: float,
    total_trees: int,
    row_spacing: float, # ระยะห่างระหว่างแถว (เมตร)
    tree_spacing: float # ระยะห่างระหว่างต้นในแถวเดียวกัน (เมตร)
) -> list[dict]:
    """
    Generates simulated UTM coordinates for rubber trees in a rectangular plot.
```

```
Args:
    center_x (float): UTM X coordinate of the center of the plot.
    center_y (float): UTM Y coordinate of the center of the plot.
    total_trees (int): Total number of trees to generate coordinates for.
    row_spacing (float): Distance between rows in meters.
    tree_spacing (float): Distance between trees within the same row in meters.

Returns:
    list[dict]: A list of dictionaries, where each dictionary represents a tree
                and contains its ID, row, column, and UTM X/Y coordinates.
"""

# Calculate approximate number of rows and columns to fit the total trees
# We aim for a roughly square plot in terms of tree count, adjusted for spacing.
# This ensures a reasonable distribution.
# Let's find num_cols and num_rows such that num_cols * num_rows >= total_trees
# And the plot dimensions are somewhat proportional to spacing.
# We can start by assuming num_cols is roughly sqrt(total_trees * (row_spacing / tree_spacing))
# And num_rows is roughly sqrt(total_trees * (tree_spacing / row_spacing))

# A simpler approach for a rectangular grid:
# Estimate a square root of total trees
approx_side = math.ceil(math.sqrt(total_trees))

# Let's try to make the number of columns and rows somewhat balanced
# We'll prioritize having enough columns to make the plot wide enough for the trees
# And enough rows to cover the remaining trees.

# Let's define the number of columns (trees per row) and rows
# We'll aim for a number of columns that makes sense for the given spacing.
# For simplicity and to ensure we generate enough trees, let's make the number of columns
# slightly more than the square root, and calculate rows from that.
```

```
# A more robust way to determine num_rows and num_cols to ensure all trees fit
# and the aspect ratio is somewhat balanced with the spacing.
# We want (num_cols * tree_spacing) / (num_rows * row_spacing) to be close to 1
# And num_cols * num_rows >= total_trees

# Let's try to make the number of columns such that the total width is reasonable.
# We will use a fixed number of columns that results in a manageable plot width.
# For 2999 trees, if we have, say, 50 trees per row, that's about 60 rows.
# 50 trees * 3m/tree = 150m width
# 60 rows * 8m/row = 480m length
# This seems reasonable. Let's adjust to ensure we cover 2999 trees.

# Let's set a target number of columns, e.g., 55 trees per row.
# This will give a width of (55-1)*3 = 162m
num_cols = 55
num_rows = math.ceil(total_trees / num_cols) # Calculate rows needed to fit all trees

print(f"Estimated number of columns (trees per row): {num_cols}")
print(f"Calculated number of rows: {num_rows}")
print(f"Total grid spots generated: {num_cols * num_rows}")

# Calculate the total dimensions of the rectangular plot
# The total length/width is based on the distance between the *first* and *last* tree/row.
# So, it's (number_of_items - 1) * spacing
plot_width = (num_cols - 1) * tree_spacing # Total width along X-axis
plot_length = (num_rows - 1) * row_spacing # Total length along Y-axis

# Calculate the starting point (bottom-left corner) of the plot
# Assuming the center_x, center_y is the true geometric center.
start_x = center_x - (plot_width / 2)
start_y = center_y - (plot_length / 2)
```

```
    tree_data = []
    tree_id_counter = 1

    # Iterate through rows and columns to generate coordinates
    for row_idx in range(num_rows):
        for col_idx in range(num_cols):
            if tree_id_counter > total_trees:
                break # Stop if we have generated enough trees

            # Calculate X and Y coordinates for the current tree
            # X increases as we move right (along columns)
            # Y increases as we move up (along rows)
            current_x = start_x + (col_idx * tree_spacing)
            current_y = start_y + (row_idx * row_spacing)

            tree_data.append({
                "Tree ID": tree_id_counter,
                "Row": row_idx + 1, # Start row number from 1
                "Column": col_idx + 1, # Start column number from 1
                "UTM_X": round(current_x, 2), # Round to 2 decimal places for readability
                "UTM_Y": round(current_y, 2)
            })
            tree_id_counter += 1
        if tree_id_counter > total_trees:
            break

    return tree_data

def save_to_csv(data: list[dict], filename: str):
    """
    Saves the generated tree data to a CSV file.
```

```
    Args:
        data (list[dict]): The list of tree data dictionaries.
        filename (str): The name of the CSV file to save.
    """
    if not data:
        print("No data to save.")
        return

    # Get header from the keys of the first dictionary
    fieldnames = data[0].keys()

    try:
        with open(filename, 'w', newline='', encoding='utf-8') as csvfile:
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader() # Write the column headers
            writer.writerows(data) # Write all the tree data rows
        print(f"Successfully saved {len(data)} tree coordinates to '{filename}'")
    except IOError as e:
        print(f"Error saving to CSV file: {e}")

# --- Configuration ---
# พิกัดศูนย์กลางของสวนยางพารา (UTM Zone 48P)
center_utm_x = 264830.0
center_utm_y = 1582035.0

# จำนวนต้นไม้ทั้งหมด
total_trees_in_plot = 1209

# ระยะห่างระหว่างแถวและระหว่างต้น (เมตร)
row_spacing_meters = 8.0
tree_spacing_meters = 3.0
```

```
# ชื่อไฟล์ CSV ที่จะบันทึก
output_csv_filename = "rubber_tree_coordinates.csv"

# --- Generate and Save ---
print("Generating rubber tree coordinates...")
tree_coordinates = generate_rubber_tree_coordinates(
    center_utm_x,
    center_utm_y,
    total_trees_in_plot,
    row_spacing_meters,
    tree_spacing_meters
)

if tree_coordinates:
    save_to_csv(tree_coordinates, output_csv_filename)
else:
    print("Failed to generate tree coordinates.")
```