RAMAIAH SKILL ACADEMY

COURSE NAME: EMBEDDED SYSTEM DESIGN

TITLE: ELECTRIC FIELD AND POTENTIAL SIMULATOR (2D VISUALIZATION) IN GITHUB

ASSIGNMENT 1

GROUP 8

KEERTHI S

NIDHI V J

SHREYA S

SNEHA B S

SWETHA G K

About GIT

Git is a distributed version control system commonly used for tracking changes in source code during software development. Here are some key points on Git:

Version Control: Git tracks the history of changes made to files in a repository. This allows developers to revert to previous versions of the code, compare changes, and collaborate effectively.

Distributed System: Unlike centralized version control systems (e.g., SVN), Git allows each user to have a full copy of the entire repository history. This decentralization improves performance and redundancy.

Branching and Merging: Git makes it easy to create branches (parallel versions of a project) to work on features or bug fixes independently. After completing the work, branches can be merged back into the main project, keeping the changes integrated.

Staging Area: Git includes a unique "staging area" where changes can be reviewed and organized before being committed. This helps in breaking down and managing the change process.

Commits: Every change in Git is stored as a commit, which is a snapshot of the file system at a particular point. Commits include metadata such as the author, timestamp, and a message describing the change.

Remote Repositories: Git supports pushing changes to and pulling from remote repositories (e.g., on platforms like GitHub, GitLab, or Bitbucket), enabling collaboration across teams.

Commands:

git init: Initializes a new Git repository.

git clone: Clones a remote repository to a local machine.

git add: Adds changes to the staging area.

git commit: Saves changes to the repository.

git pull: Fetches changes from a remote repository and merges them.

git push: Pushes local commits to a remote repository.

Conflict Resolution: When multiple changes affect the same part of the code, Git helps resolve conflicts by allowing the user to manually merge the changes.

Collaboration: Teams can work on different branches and use pull requests (PRs) to review and merge code, fostering collaborative development practices.

These features make Git a powerful tool for maintaining code quality, collaboration, and tracking progress in software development projects.

About GITHUB

GitHub is a powerful platform for version control and collaboration, commonly used for managing code and project documentation.

Key Features on GitHub:

Version Control: GitHub keeps track of all changes made to the report, allowing you to review, revert, or compare different versions easily. Multiple collaborators can work on the same report without overwriting each other's work.

Markdown Support: You can write and format reports directly in GitHub using Markdown, a lightweight markup language. Markdown allows you to create headings, bullet points, links, tables, images, and even code blocks in a clean, readable format.

Collaboration: GitHub facilitates collaboration with team members via Pull Requests. This allows you to propose changes and discuss them before merging into the main document. Issues and Discussions enable you to track tasks, receive feedback, and address specific sections of the report.

Hosting and Sharing: You can use GitHub Pages to host and share your report in a publicly accessible format, which is useful if you want to present the report as a website. GitHub also provides a direct link to the raw document for easy sharing.

Security and Access Control: GitHub allows you to manage who has access to the report with public or private repositories. You can control who can view or edit the report.

Integration with Other Tools: GitHub integrates with tools like Microsoft Word, Google Docs, or LaTeX for more advanced document editing. GitHub also supports linking with continuous integration tools, making it useful for technical reports that include automated data or code. Using GitHub for reports ensures collaboration, transparency, and ease of managing versions, making it an ideal tool for both technical and non-technical documents.

Steps to install Git & GitHub on Windows

1. Install Git:

Git is essential for interacting with GitHub repositories.

Download Git:

Visit the Git for Windows website and download the latest version.

Install Git:

Run the downloaded .exe file.

Follow the installation prompts. You can leave most options as default unless you have specific preferences.

Configure Git:

Open a terminal (e.g., Command Prompt, PowerShell) and set your Git username and email:

```
git config --global user.name "Your Name"

git config --global user.email "youremail@example.com"
```

2. Install GitHub Desktop (Optional):

GitHub Desktop provides a GUI interface for managing Git repositories.

Download GitHub Desktop:

Visit the GitHub Desktop website and download the latest version.

Install GitHub Desktop:

Run the installer and follow the prompts.

Login to GitHub:

Open GitHub Desktop and sign in with your GitHub account.

3. Using Git in Command Line (Optional):

You can clone repositories and push changes via Git commands.

Clone a repository:

```
git clone https://github.com/username/repository.git
```

Push changes to GitHub:

```
git add .

git commit -m "Your commit message"

git push origin main
```

After this setup, you should be able to use Git and GitHub on your Windows machine.

1. Installation Instructions

Before you can use the visualization tool, you need to ensure it is properly installed. Follow these steps:

Prerequisites

Ensure you have Python installed (preferably Python 3.6 or later).

Install necessary libraries, which may include NumPy and Matplotlib.

Installation Steps

1. Clone the Repository or Download the Package:

If it's available on a platform like GitHub, you can clone it using:

```
git clone https://github.com/username/repo-name.git

cd repo-name
```

Alternatively, download the zip file and extract it.

2. Install Required Libraries: You can use pip to install the required libraries. Open your terminal or command prompt and run:

```
pip install numpy matplotlib
```

3. Run the Tool: Navigate to the directory where the tool is located and run:

```
python visualization_tool.py
```

2. Theoretical Background

Electric Field

The electric field E around a charge is a vector field that represents the force per unit charge exerted on a positive test charge placed in the field. It is defined as:

$$E=F/q$$

Where, F is the force experienced by the test charge q.

For a point charge Q, the electric field at a distance r is given by:

$$E=k\cdot|Q| / r^2$$

Where, k is Coulomb's constant (8.99×10^9 Nm^2/C^2).

Electric Potential

The electric potential V at a point in an electric field is the work done in bringing a unit positive charge from infinity to that point, without any acceleration. It is given by:

$$V=k\cdot Q/r$$

For multiple point charges, the total potential is the sum of the potentials due to each charge.

3. Function Explanations

3.1. Setting Up the Environment

- Function: setup_environment()
- Description: Initializes the plot area and settings for the visualization.
- Usage: setup_environment()

3.2. Plotting Electric Fields

- Function: plot_electric_field(charges, grid_size)
- Parameters: charges: A list of tuples representing charge positions and magnitudes, e.g., [(1, (0, 0)), (-1, (1, 1))].

  grid_size: Tuple indicating the size of the grid for the field, e.g., (10, 10).

  Description: Calculates and visualizes the electric field generated by the specified charges on the grid.

Usage: plot_electric_field([(1, (0, 0)), (-1, (1, 1))], (10, 10))

3.3. Plotting Electric Potential

- Function: plot_potential(charges, grid_size)
- Parameters: Same as plot_electric_field.
- Description: Calculates and visualizes the electric potential generated by the specified charges on the grid.
- Usage: plot_potential([(1, (0, 0)), (-1, (1, 1))], (10, 10))

4. Usage Examples

Example 1: Visualizing the Electric Field

```
from visualization_tool import setup_environment, plot_electric_field

setup_environment()

plot_electric_field([(1, (0, 0)), (-1, (1, 1))], (10, 10))
```

This example sets up the environment and visualizes the electric field due to a positive charge at the origin and a negative charge at (1, 1).

Example 2: Visualizing Electric Potential

```
from visualization_tool import setup_environment, plot_potential

setup_environment()

plot_potential([(1, (0, 0)), (-1, (1, 1))], (10, 10))
```

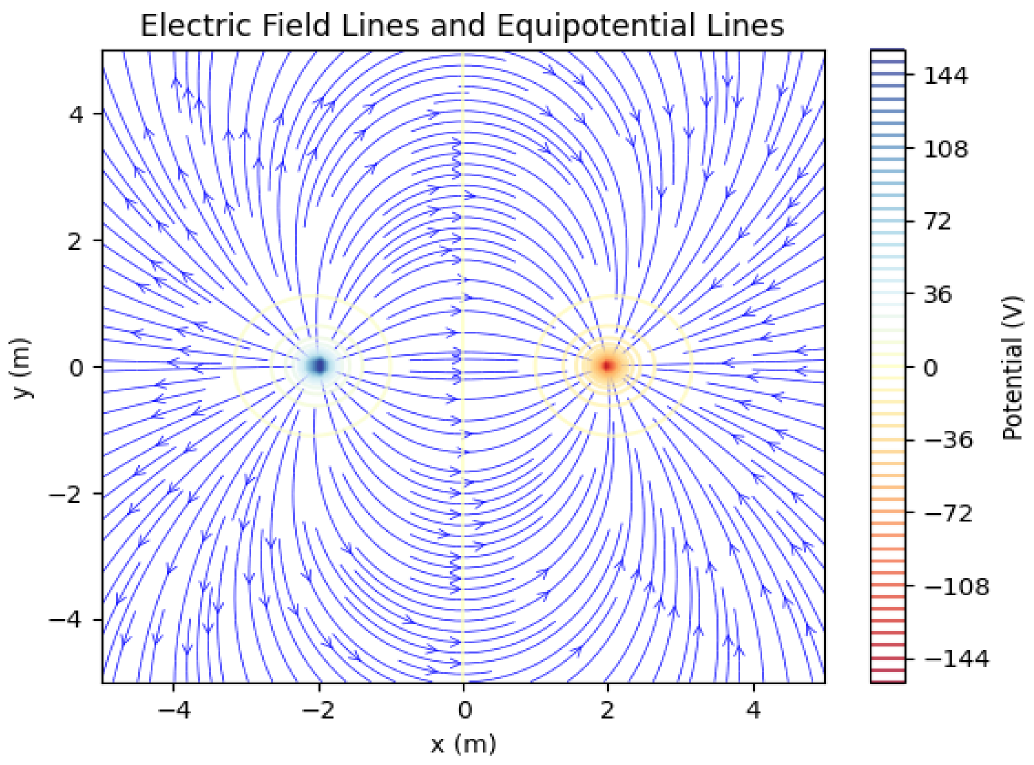
This example visualizes the electric potential due to the same charges.

CODE

```
import numpy as np
import matplotlib.pyplot as plt

# Constants
k = 8.99e9  # Coulomb's constant in N m²/C²

# Define a class for a point charge
class PointCharge:
    def __init__(self, q, position):
        self.q = q  # Charge in Coulombs
        self.position = np.array(position)  # Position as an (x, y)
tuple

    def electric_field(self, x, y):
        """
        Calculate the electric field vector at a point (x, y) due to
this charge.
        E = k * q * r / |r|^3
        """
        r = np.array([x, y]) - self.position  # Position vector
relative to charge
        r_magnitude = np.linalg.norm(r)  # Distance from charge to
point
        if r_magnitude == 0:
            return np.array([0, 0])  # Avoid division by zero at
charge location
        return k * self.q * r / r_magnitude**3

    def potential(self, x, y):
        """
        Calculate the electric potential at a point (x, y) due to this
charge.
        V = k * q / |r|
        """
        r = np.array([x, y]) - self.position
        r_magnitude = np.linalg.norm(r)
        if r_magnitude == 0:
            return 0  # Avoid division by zero
        return k * self.q / r_magnitude
```

```
# Define a function to calculate the total electric field due to
multiple charges
def total_electric_field(charges, x, y):
    """
    Calculate the total electric field at a point (x, y) due to all
charges.
    """
    E_total = np.array([0.0, 0.0])
    for charge in charges:
        E_total += charge.electric_field(x, y)
    return E_total

# Define a function to calculate the total potential due to multiple
charges
def total_potential(charges, x, y):
    """
    Calculate the total electric potential at a point (x, y) due to
all charges.
    """
    V_total = 0.0
    for charge in charges:
        V_total += charge.potential(x, y)
    return V_total

# Visualization Functions (Already Provided)

def plot_electric_field_lines(charges, xlim=(-5, 5), ylim=(-5, 5),
grid_size=100):
    """
    Plot electric field lines due to multiple charges.
    """
    x = np.linspace(xlim[0], xlim[1], grid_size)
    y = np.linspace(ylim[0], ylim[1], grid_size)
    X, Y = np.meshgrid(x, y)

    Ex, Ey = np.zeros(X.shape), np.zeros(Y.shape)

    # Calculate electric field at each grid point
    for i in range(grid_size):
        for j in range(grid_size):
            E = total_electric_field(charges, X[i, j], Y[i, j])
            Ex[i, j], Ey[i, j] = E

    plt.streamplot(X, Y, Ex, Ey, color='blue', linewidth=0.5,
density=2, arrowstyle='->')
```

```
    plt.title('Electric Field Lines')
    plt.xlabel('x (m)')
    plt.ylabel('y (m)')
    plt.show()

def plot_equipotential_lines(charges, xlim=(-5, 5), ylim=(-5, 5),
grid_size=100):
    """
    Plot equipotential lines due to multiple charges.
    """
    x = np.linspace(xlim[0], xlim[1], grid_size)
    y = np.linspace(ylim[0], ylim[1], grid_size)
    X, Y = np.meshgrid(x, y)

    V = np.zeros(X.shape)

    # Calculate potential at each grid point
    for i in range(grid_size):
        for j in range(grid_size):
            V[i, j] = total_potential(charges, X[i, j], Y[i, j])

    plt.contour(X, Y, V, levels=50, cmap='RdYlBu')
    plt.title('Equipotential Lines')
    plt.xlabel('x (m)')
    plt.ylabel('y (m)')
    plt.colorbar(label='Potential (V)')
    plt.show()

def combined_plot(charges, xlim=(-5, 5), ylim=(-5, 5), grid_size=100):
    """
    Combine electric field lines and equipotential lines into one
plot.
    """
    x = np.linspace(xlim[0], xlim[1], grid_size)
    y = np.linspace(ylim[0], ylim[1], grid_size)
    X, Y = np.meshgrid(x, y)

    Ex, Ey = np.zeros(X.shape), np.zeros(Y.shape)
    V = np.zeros(X.shape)

    # Calculate both electric field and potential at each grid point
    for i in range(grid_size):
        for j in range(grid_size):
            E = total_electric_field(charges, X[i, j], Y[i, j])
            Ex[i, j], Ey[i, j] = E
            V[i, j] = total_potential(charges, X[i, j], Y[i, j])

    plt.streamplot(X, Y, Ex, Ey, color='blue', linewidth=0.5,
density=2, arrowstyle='->')
    plt.contour(X, Y, V, levels=50, cmap='RdYlBu', alpha=0.75)
    plt.title('Electric Field Lines and Equipotential Lines')
    plt.xlabel('x (m)')
    plt.ylabel('y (m)')
    plt.colorbar(label='Potential (V)')
    plt.show()

# Example usage
if __name__ == "__main__":
    charge1 = PointCharge(1e-9, (-2, 0))  # +1nC charge at (-2, 0)
    charge2 = PointCharge(-1e-9, (2, 0))  # -1nC charge at (2, 0)

    charges = [charge1, charge2]

    combined_plot(charges)
```


Electric Field Lines and Equipotential Lines
y (m)
x (m)
Potential (V)