**Project Documentation: Python-Based Agricultural Soil-Rover Simulation**

**1. Introduction**

This project transitions the MATLAB-based agricultural simulation into a real-time, interactive Python framework using Pygame and Mayavi. It aims to model the interaction between a robotic rover and deformable terrain represented by granular particles. The simulation explores soil displacement, rover mobility, and terrain modeling through particle physics — establishing a scalable base for precision agriculture applications using only Python-native tools.

**2. Objectives**

* Simulate realistic farmland terrain using noise and layered particle-based elevation models
* Implement granular soil behavior using particle mechanics and custom physics rules
* Design a movable rover capable of interacting with and deforming the terrain
* Optimize real-time rendering and collision performance using NumPy and Numba
* Visualize 3D terrain, rover motion, and particle response using Mayavi
* Build a modular Python framework adaptable for sensor integration and future ROS2 migration

**3. Simulation Environment Setup**

**Terrain Generation**

* A 10×10 meter terrain grid was created using sinusoidal wave functions and Gaussian noise overlays
* Terrain smoothing was applied using scipy.ndimage.gaussian\_filter to remove sharp spikes
* Bumps were defined via configurable parameters for amplitude, width, and frequency
* NumPy arrays enabled efficient manipulation of height values and terrain reshaping

**Particle-Based Terrain Representation**

* Terrain height data was converted into vertical particle stacks with 0.02m diameter grains
* Each particle had properties for position, velocity, and mass, stored in structured arrays
* Visualization used mlab.points3d() in Mayavi with per-layer coloring and dynamic lighting
* Grid spacing was set to 0.05m for detail-performance balance

**4. Rover Modeling and Physics Integration**

**Rover Class**

* The rover was represented using a custom Rover class containing position, velocity, and size
* A draw\_custom\_rover() function visualized the rover as a 3D cuboid (later upgraded to cylindrical)
* Dimensions: 0.3×0.3×0.15 meters, with color and opacity settings for clear distinction

**Motion & Interaction**

* Keyboard controls enabled rover translation across the terrain
* The rover’s height updated dynamically based on terrain Z-values at its (x, y) position
* Collision detection with soil particles initiated displacement and momentum propagation
* Rover-particle interactions applied force vectors to nearby particles simulating plowing

**5. Terrain Modification and Particle Physics**

**Granular Mechanics**

* Particles obeyed simple rules: fall straight down, slide diagonally if blocked
* Frictional forces were modeled between vertical particle layers based on relative velocity
* Numba-optimized functions accelerated per-frame calculations to allow >10k particles

**Soil Deformation**

* When the rover moved through the terrain, particles were displaced in its path
* Upper layers transferred force to lower layers, simulating compression and displacement
* Track marks and furrowed trails emerged dynamically from repeated rover passes

**6. Visualization and Real-Time Feedback**

* Combined Mayavi 3D visualization with Matplotlib for optional heatmaps or statistics
* Used layered color maps to indicate elevation, particle density, or deformation zones
* Real-time animation loop updated both rover and terrain at 60 FPS
* Enabled scene rotation, zoom, and slicing to observe subsurface changes

**7. Challenges Encountered**

**Particle Rendering Bottleneck**

* Rendering millions of particles caused major slowdowns
* Solved by subsampling particles for display and optimizing logic with Numba

**Terrain-Particle Alignment**

* Initial mismatches between terrain height and particle placement were fixed by thresholding placement to Z[i,j] > min\_height

**Persistent Object Updates**

* Avoided full-scene redrawing by keeping persistent Mayavi handles and using .mlab\_source.set() for updates

**8. Key Outcomes**

* Successfully transitioned from MATLAB to a Python-only simulation pipeline for soil mechanics
* Built a granular terrain environment that responds to rover motion and interaction
* Achieved visually compelling and physically interactive results without external physics engines
* Established a performant and modular base for integrating autonomous navigation or sensor feedback

**9. Future Extensions**

**Sensor & SLAM Integration**

* Add virtual GNSS and camera sensors to simulate sensor-driven navigation
* Implement EKF to fuse IMU and visual odometry with rover position

**ROS2 Migration**

* Port core rover and particle logic into ROS2 nodes with rclpy, nav\_msgs, and sensor\_msgs
* Use Gazebo or Ignition for realistic world models and controller interfaces

**AI-Based Control**

* Use reinforcement learning to optimize plowing paths or minimize soil compaction
* Integrate ML-based terrain classification to dynamically adjust rover behavior

**Complex Terrain**

* Introduce multi-layered soils with different particle types and behaviors
* Simulate weathering, water seepage, or root structures for biological realism

**10. Conclusion**

This Python-based simulation advances virtual agricultural robotics by leveraging lightweight, open-source tools for high-fidelity soil modeling and rover control. The integration of real-time particle dynamics, interactive terrain modification, and modular design showcases the potential for accessible, scalable farm robotics prototyping. It lays a strong foundation for transitioning to full robotic system deployment in Gazebo, ROS2, or real hardware environments.

**Author**: Adharsh Prasad Natesan  
**Project Period**: Fall 2024 – Spring 2025  
**Environment**: Python 3.10, Mayavi, NumPy, Pygame, Numba  
**Repository**: [Add GitHub URL here if applicable]