MANE 4220 – Inventor's Studio II
Final Design Report

Deformation-Displacement Array (DDA) Sensor

Aaryan Sonawane & Hannah Fried

1. Problem, Solution & Customer

Our group has identified a gap in the market for customized orthopedic solutions between the choice of prefabricated, cheap, but sub-optimally fitted orthotics from pharmaceutical scanners & kits, and expensive, slow-to-obtain, better fitting options gotten from specialists. In searching for potential applications for our new type of sensor - the Deformation Displacement Array sensor – we found a way to improve on the process of taking data for patients' orthopedic needs by taking dynamic geometric data of their weight pressure profiles, in addition to their pressure distributions when standing. Additionally, our device possesses the ability to generate usable 3D geometry in the form of STL files – which are 3D printable and can allow for rapid manufacturing of orthotic components on faster timescales than in-office specialists (Hours to days, compared to days or weeks.)

The Deformation-Displacement Array (DDA) Sensor aims to provide an innovative solution for digitally capturing and analyzing 3D shapes of deformable objects. Using an array of pins equipped with sensors, the DDA sensor measures the displacement of each pin when subjected to external forces. By combining the DDA sensor with the LIDAR 3D scanner app available on iPhones, this project enhances object profiling by leveraging both precise local displacement data and external geometry mapping. This combined approach enables the creation of a comprehensive digital map of an object's external 3D surface in real time. Traditional methods of capturing object profiles rely primarily on 3D scanning or external imaging devices, which are often limited in their ability to capture intricate surface deformations. In contrast, the DDA sensor's approach provides an advanced solution for accurately mapping the complex geometries of deformable objects' external surfaces in real time.

There are two main "customers" for our solution: the patients themselves with foot or walking problems which necessitate orthopedic help, and the doctors & pharmacists trying to connect patients with solutions. Our device, upon deployment in the real world, would ideally be a component of a kiosk with a footprint inside pharmacies; this is akin to *Dr. Scholls* machines already common across the US, which dispense prefabricated orthotics. Alternatively, having brick & mortar locations to allow customers to visit key cities and have their data taken could also work. An ideal patient would have an immediate need for orthopedic assistance, i.e. a

maternity patient experiencing foot problems from rapid weight gain, but no severe or unusual medical complications, and preferably an endorsement from their doctor. They may also have financial constraints which the cheap & rapid construction of orthotics that our sensor facilitates could help with.

There are other non-medical customers which our sensor may be able to serve too as a piece of lab equipment, though our primary exploration in this class was orthopedics focused. Other scientific applications are less brainstormed as of now, while the medical applications are apparent & potentially lucrative.

Target Users for the Sensor

- Pharmacy employees (Orthotic fitting)
- Medical professionals (Prosthetic fitting)
- Disabled individuals (Controllers and interaction devices)
- Scientists (For mapping of internal structure of deformable bodies)

Current Market Solutions and Limitations

1. Traditional 3D Scanners
    a. High cost ($5,000+) [1]
    b. Limited real-time capability [2]
    c. Poor performance with deformable surfaces [3]
2. Vision-based Systems
    a. Affected by lighting conditions [4]
    b. Cannot capture occluded surfaces [5]
    c. Limited accuracy for small deformations [6]
3. Contact-based Measurement Tools
    a. Time-consuming manual operation [7]
    b. Limited measurement points [8]
    c. No real-time data processing [9]

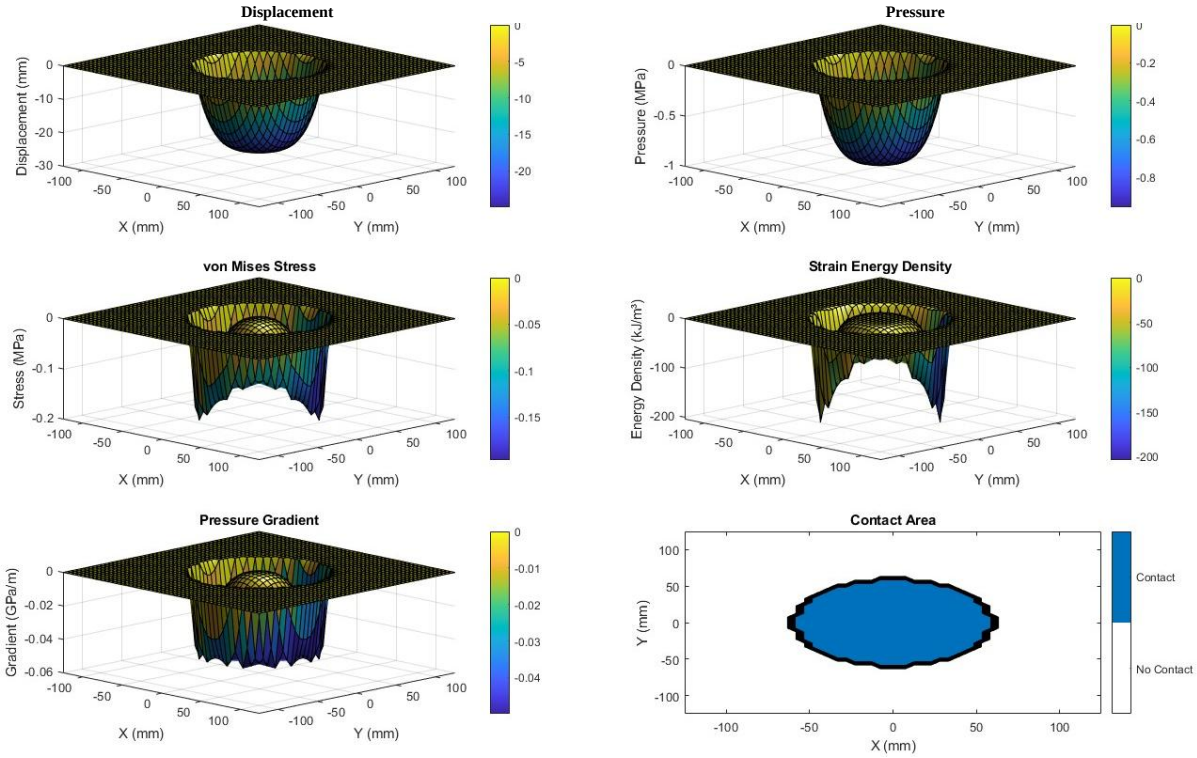
2. Project Goals and Deliverables

2.1 Software Subsystem

As we prepared our prototype for operation, we realized that the data collected solely from the prototype was unusable due to the limited number of pins on the sensors. This led us to pivot towards creating a more visual proof of concept. We realized that the DDA sensor alone was not sustainable as a standalone product but would work better as part of a larger assembly. To address this, we used extensive modeling software, including Houdini, MATLAB, and Python,

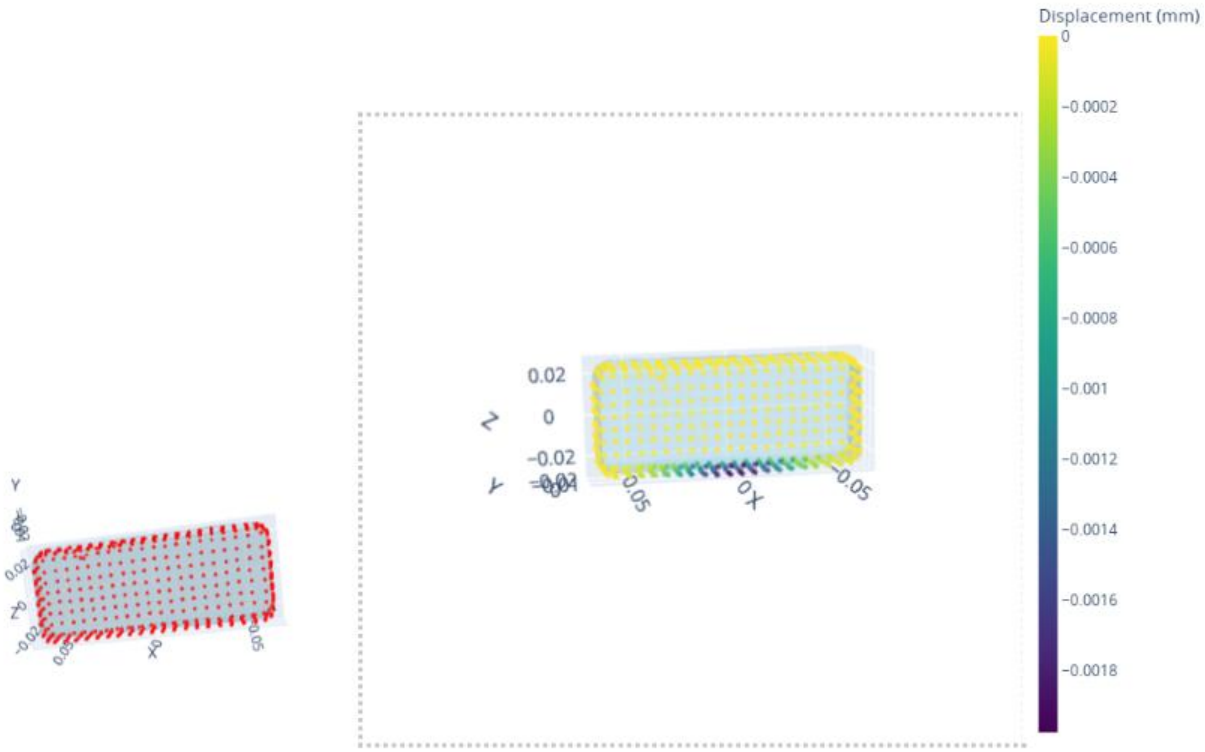
to map displacement metrics and create visualizations. This approach also helped illustrate how a larger pin array could function.

Figure 2.1 Deformation metrics of a simulated stuffed animal on a DDA array



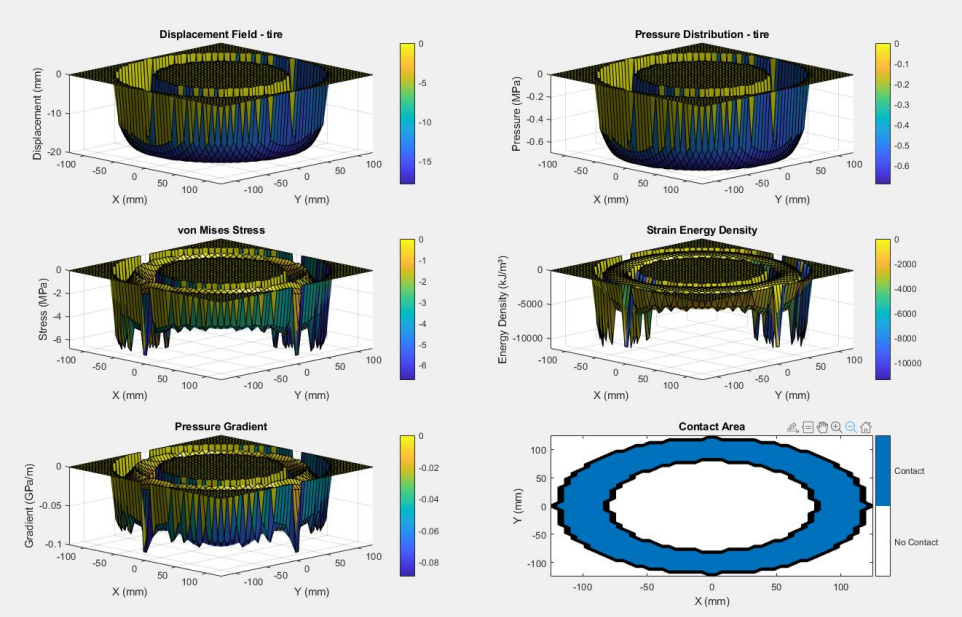
This plot differs from the previously submitted one because it simulates the real prototype and focuses on integrating it with the assembly. The data used to form the mesh enclosing the point cloud in Figure 2.2 was sourced from the iPhone 3D scanner. Since MATLAB couldn’t directly connect to the scanner data, only one face of custom objects was used for the MATLAB plots. Despite this limitation, MATLAB's visual and computational efficiency proved more useful for the project compared to Python.

Figure 2.2 Minute deformation comparison for a foam eraser from the IS2 Lab



Due to the sheer intensity of some of the DDA sensor array designs visually, Matlab was chosen to the best medium to perform FEA Analysis. The displacement, Pressure gradient, von mises stress, strain and contact areas were plotted in Matlab using sample data. Matlab's advanced visualization and analytical tools provided the precision and clarity needed to effectively interpret these metrics

Figure 2.3 FEA Analysis of a sample tire placed on top of a DDA array



As displayed in class, we also met up with VFX studio professor Phil Vanderhyden who helped us incorporate a stunning visual of a detailed DDA array as shown in our presentation. Initially, we faced challenges in visualizing the data due to its sheer size. Additionally, we encountered licensing limitations with the Houdini software, restricting our ability to fully utilize its features. Despite these hurdles, the expertise of Prof Vanderhyden helped us overcome these obstacles, resulting in a compelling and visually engaging depiction of our concept.

2.2 Hardware Subsystem

Figure 2.4 The Pin Array of the DDA Sensor Prototype, 3D-Printed & Assembled

Our hardware subsystem – originally the primary focus of our efforts in this project, is an array of 16 (Originally 18, but downsized due to hardware constraints) pins in a lattice. The pins are connected to linear displacement sensors, in this case potentiometers, which are connected to a microcontroller visible on the bottom shelf of the housing. The pins are mounted on top of compression springs to facilitate linear displacement resistance. The analog voltage outputs of the potentiometers are connected to a microcontroller, which is outfitted with SD-card writing capability to log spatial data and transfer it to a computer for further processing.

The springs are off-the-shelf components, as well as the structural components of the housing such as nuts, bolts, etc. All off-the-shelf components have costs below 3$ per unit, though summed together their costs approached roughly 40$. Most of the load-bearing housing & paneling is machined or 3D-Printed using our group's own resources, making them free save for time & labor to produce the parts. These items include the face and base plates, pistons, and feet. Additionally, our microcontroller & SD-card-writer are components from Italian prototyping company *Arduino,* which cost about 40$ new, but our group members owned these circuit boards already.

The hardware prototype, while capable of transmitting data and taking deformation readings, was ultimately little more than a showpiece demonstrating that the sensor is technically possible; this was even already proven – we have not invented a novel type of displacement detection, and placing sensors in an array is a design choice, but not an innovation in and of itself. Plus, an array of only 16 usable data points is not to scale of any real-life prototype, which would likely house hundreds if not thousands of displacement pins of "resolution." The constructed prototype is, at least, roughly the size of an ideal final device, even if the pin resolution is poor, so it remains useful for visual communication.

Figure 2.5 The Microcontroller, and the SD-card writer attached to its top surface

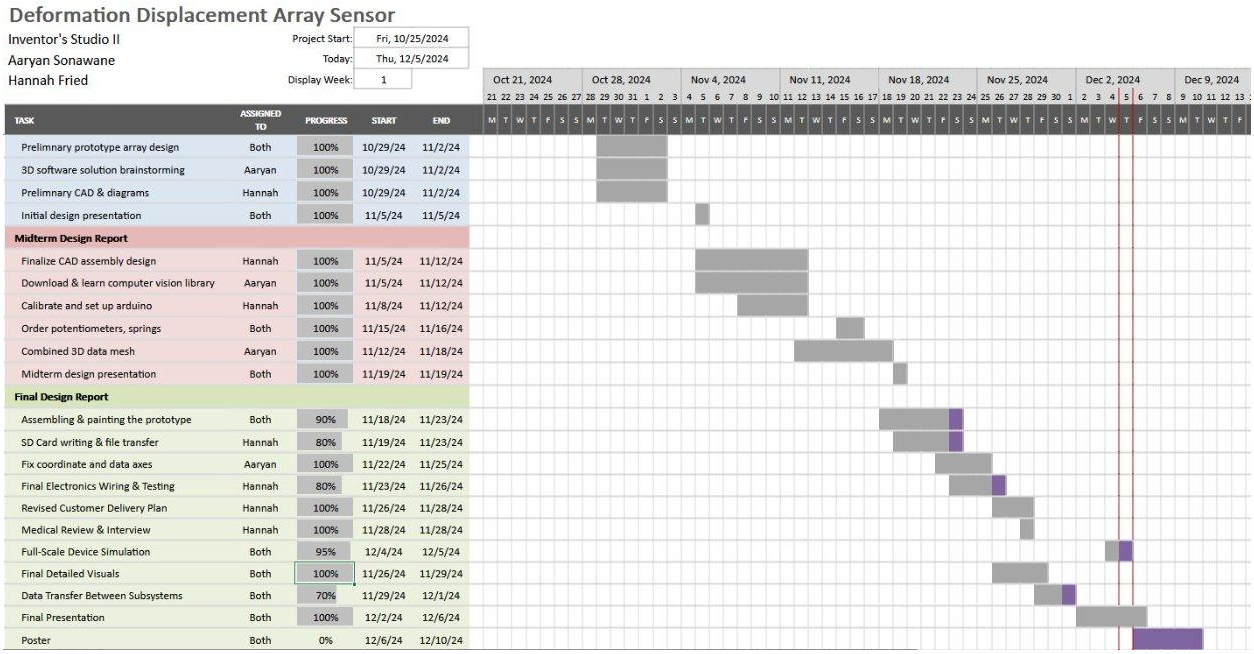
3. Technology Readiness Level (TRL)

- Starting TRL: 1 (Experimental proof of concept)

- Target TRL: 3 (Technology validated in relevant environment)
- Key Advancement: Implementation of integrated hardware-software system

4. Gantt Chart

Figure 4.1 Gantt Chart for the DDA Sensor



Deformation Displacement Array Sensor

Inventor's Studio II | Project Start: Fri, 10/25/2024
Aaryan Sonawane | Today: Thu, 12/5/2024
Hannah Fried | Display Week: 1

TASK	ASSIGNED TO	PROGRESS	START	END
Prelimnary prototype array design	Both	100%	10/29/24	11/2/24
3D software solution brainstorming	Aaryan	100%	10/29/24	11/2/24
Prelimnary CAD & diagrams	Hannah	100%	10/29/24	11/2/24
Initial design presentation	Both	100%	11/5/24	11/5/24
Midterm Design Report				
Finalize CAD assembly design	Hannah	100%	11/5/24	11/12/24
Download & learn computer vision library	Aaryan	100%	11/5/24	11/12/24
Calibrate and set up arduino	Hannah	100%	11/8/24	11/12/24
Order potentiometers, springs	Both	100%	11/15/24	11/16/24
Combined 3D data mesh	Aaryan	100%	11/12/24	11/18/24
Midterm design presentation	Both	100%	11/19/24	11/19/24
Final Design Report				
Assembling & painting the prototype	Both	90%	11/18/24	11/23/24
SD Card writing & file transfer	Hannah	80%	11/19/24	11/23/24
Fix coordinate and data axes	Aaryan	100%	11/22/24	11/25/24
Final Electronics Wiring & Testing	Hannah	80%	11/23/24	11/26/24
Revised Customer Delivery Plan	Hannah	100%	11/26/24	11/28/24
Medical Review & Interview	Hannah	100%	11/28/24	11/28/24
Full-Scale Device Simulation	Both	95%	12/4/24	12/5/24
Final Detailed Visuals	Both	100%	11/26/24	11/29/24
Data Transfer Between Subsystems	Both	70%	11/29/24	12/1/24
Final Presentation	Both	100%	12/2/24	12/6/24
Poster	Both	0%	12/6/24	12/10/24

The important takeaway from the Gantt chart is our pivot towards the end. As we realized that gluing together the hardware and constant testing won't help forming a proof of working concept for the project since the data being used would be minimal and not useful for any analysis. Hence, we switched to a pure digital approach to extend proof of working idea as the semester comes to a close.

5. Endpoint & Deviations

The goals of the DDA sensor evolved significantly from its original concept. The shift to a more digital/CGI form of simulation enhanced the proof of concept. The project evolved from trying to be squeezed into a product to finding a long-term use case as a component sensor. Using Matlab, the need for the iPhone 3d scanner was also taken out, giving more detailed images in one axis, sacrificing the 3d capabilities of the original plots. Our priorities shifted heavily over time from simply trying to prove the technical viability of the sensor, and then trying to shoehorn it into as many solutions as we could identify, to more carefully considering its role on broader human-oriented ecosystems to solve problems and help as many people as possible.

6. Lessons Learned

Through the course of the project, the team learned several key lessons:

- Balancing Priorities: The team learned the importance of prioritizing quality data and analysis over rapid physical prototyping. Software simulation often provides superior insights.
- Problem-Solving capabilities: Hardware limitations inspired creative solutions, such as integrating iPhone 3d scanning and leveraging simulations to validate concepts.
- Refining Market Fit: Collaborations and consultations helped refine the focus from a working product to an orthopedic solution while uncovering additional market potential.
- Adaptability: The ability to pivot and integrate feedback into project development was crucial in achieving a robust, multifunctional endpoint.

7. Future Work

Towards the end of our project's lifetime, it took on a very collaborative nature as we undertook professional consultations and began speaking with doctors about field applications for our device. The institution which took the most interest was the University of Pittsburgh Medical Center, of which a resident doctor expressed willingness to assist in bringing our sensor into an orthopedic solution ecosystem on-market.

To facilitate this cooperation and bring the idea to fruition, we intend to file for intellectual property rights, most likely a design patent, to protect our technical advantage from other competitors in the space who might also specialize in pharmaceutical footprint solutions, or direct home delivery kits, like *Dr Scholls* or *Corefit* Orthotics. Additionally, we may need to pursue FDA approval and likely undergo lots of revisions and clinical testing before receiving the rubber stamp to allow public interaction with our device.

We may require significant upfront investment from both RPI and UPMC financially, and the timeline to bring this product to market may take many years, but the benefits seem lucrative and broadly applicable enough to both ourselves and our consulting specialists, that the effort is likely worth it. The project as it exists now does not resemble the result we originally envisioned, but we are proud of our work, and wish to see it through to help in the real world.

8. References

[1] "3D Laser Scanning Limitations," Engineers Edge. Available: https://www.engineersedge.com/inspection/3d_laser_scanning_limitations.htm

[2] "Exploring 3D Scanning Applications and Limitations," Griffin Industries. Available: https://griffinindustries.com/the-applications-and-limitations-of-3d-laser-scanning/

[3] "3D Scanners: How They Work and Which One is Right for You," MatterHackers. Available: https://www.matterhackers.com/about/3d-scanners-how-they-work-and-which-one-is-right-for-you

[4] "Common Challenges of 3D Scanning and How to Overcome Them," M2 Prototype. Available: https://www.m2prototype.com/common-challenges-of-3d-scanning-and-how-to-overcome-them/

[5] "Types of 3D Scanning Technology: Pros & Cons," Crow Engineering. Available: https://crowengineering.com/engineering-design-services/types-of-3d-scanning-technology-pros-cons/

[6] "The Basics of 3D Scanning: An Introduction to the Technology and Its Applications," peel 3d. Available: https://store.peel-3d.com/blog/the-basics-of-3d-scanning-an-introduction-to-the-technology-and-its-applications/

[7] "Advantages & Disadvantages of 3D Laser Scanning," 3 Space. Available: https://3space.com/advantages-disadvantages-of-3d-laser-scanning/

[8] "Advantages vs Disadvantages of 3D Laser Scanner," Fibrox3D. Available: https://www.fibrox3d.com/advantages-vs-disadvantages-of-3d-laser-scanner/

[9] "3D Laser Scanning in Architecture and Construction," TrueScan. Available: https://truescan3d.com/pros-and-cons-of-3d-scanning-in-architecture-and-construction/

8. Code Appendix

8.1 SensorFEA.m

```
function main()
% List of available test cases
testCases = {'cylinder', 'hemisphere', 'finger', 'gripper', ...
'foot', 'stuffed_animal', 'tire', 'pen', 'phone', ...
'keyboard_key', 'tennis_ball'};
% Run analysis for each test case
for i = 1:length(testCases)
fprintf('\n\nRunning analysis for test case %d/%d: %s\n', ...
i, length(testCases), upper(testCases{i}));
```

```
fprintf('========================================\n');
analyzeSensorDeformation(testCases{i});
pause(1); % Pause to allow viewing of results
end
end
function analyzeSensorDeformation(objectType)
if nargin < 1
objectType = 'cylinder';
end
% Sensor array properties
sensorSpacing = 0.005;
arraySize = [50, 50];
sensorStiffness = 1000;
% Create position matrices
[X, Y] = meshgrid(linspace(-0.125, 0.125, arraySize(1)), ...
linspace(-0.125, 0.125, arraySize(2)));
% Get object parameters
objectParams = getObjectParameters(objectType);
% Calculate responses
[displacements, forces, pressures] = calculateResponse(X, Y, objectType,
objectParams, sensorStiffness);
% Calculate additional metrics
[vonMises, strainEnergy, pressureGrad] = calculateMetrics(X, Y, displacements,
pressures, objectParams);
% Visualize results
plotResults(X, Y, displacements, forces, pressures, vonMises, strainEnergy,
pressureGrad, objectType, objectParams);
end
function params = getObjectParameters(objectType)
% Base material properties
steel = struct('E', 200e9, 'nu', 0.3);
rubber = struct('E', 2e6, 'nu', 0.49);
plastic = struct('E', 2e9, 'nu', 0.35);
soft_tissue = struct('E', 15e6, 'nu', 0.45);
plush = struct('E', 1e5, 'nu', 0.45);
switch lower(objectType)
case 'cylinder'
params = struct('radius', 0.05, 'mass', 2.0, 'maxDisp', 0.01, ...
'E', steel.E, 'nu', steel.nu, 'shape', 'cylinder');
case 'hemisphere'
params = struct('radius', 0.04, 'mass', 1.5, 'maxDisp', 0.015, ...
'E', rubber.E, 'nu', rubber.nu, 'shape', 'sphere');
case 'finger'
params = struct('width', 0.02, 'length', 0.04, 'mass', 0.5, ...
'maxDisp', 0.008, 'E', soft_tissue.E, 'nu', soft_tissue.nu, ...
'shape', 'elliptical');
case 'gripper'
```

```
params = struct('width', 0.03, 'length', 0.06, 'mass', 1.0, ...
'maxDisp', 0.012, 'E', rubber.E, 'nu', rubber.nu, ...
'shape', 'rectangular');
case 'foot'
params = struct('width', 0.08, 'length', 0.25, 'mass', 25.0, ...
'maxDisp', 0.020, 'E', soft_tissue.E, 'nu', soft_tissue.nu, ...
'shape', 'foot');
case 'stuffed_animal'
params = struct('radius', 0.06, 'mass', 0.3, 'maxDisp', 0.025, ...
'E', plush.E, 'nu', plush.nu, 'shape', 'soft_sphere');
case 'tire'
params = struct('width', 0.05, 'radius', 0.1, 'mass', 5.0, ...
'maxDisp', 0.018, 'E', rubber.E, 'nu', rubber.nu, ...
'shape', 'tire');
case 'pen'
params = struct('width', 0.008, 'length', 0.12, 'mass', 0.02, ...
'maxDisp', 0.005, 'E', plastic.E, 'nu', plastic.nu, ...
'shape', 'cylinder');
case 'phone'
params = struct('width', 0.07, 'length', 0.15, 'mass', 0.2, ...
'maxDisp', 0.006, 'E', plastic.E, 'nu', plastic.nu, ...
'shape', 'rectangular');
case 'keyboard_key'
params = struct('width', 0.015, 'length', 0.015, 'mass', 0.005, ...
'maxDisp', 0.004, 'E', plastic.E, 'nu', plastic.nu, ...
'shape', 'square');
case 'tennis_ball'
params = struct('radius', 0.033, 'mass', 0.057, 'maxDisp', 0.010, ...
'E', rubber.E, 'nu', rubber.nu, 'shape', 'sphere');
end
end
function [displacements, forces, pressures] = calculateResponse(X, Y, objectType,
params, k)
displacements = zeros(size(X));
R = sqrt(X.^2 + Y.^2);
switch params.shape
case 'cylinder'
edge_region = 0.8 * params.radius;
core = R <= edge_region;
displacements(core) = params.maxDisp;
transition = R > edge_region & R <= params.radius;
displacements(transition) = params.maxDisp * ...
cos(pi/2 * (R(transition) - edge_region)/(params.radius - edge_region));
case 'sphere'
contact = R <= params.radius;
displacements(contact) = params.maxDisp * ...
sqrt(1 - (R(contact)/params.radius).^2);
```

```
case 'soft_sphere'
contact = R <= params.radius;
displacements(contact) = params.maxDisp * ...
(1 - (R(contact)/params.radius).^4);
case {'elliptical', 'foot'}
X_norm = X/params.length;
Y_norm = Y/params.width;
R_ellip = sqrt(X_norm.^2 + Y_norm.^2);
contact = R_ellip <= 1;
displacements(contact) = params.maxDisp * ...
(1 - R_ellip(contact).^2).^1.5;
case {'rectangular', 'square'}
X_norm = abs(X/params.length);
Y_norm = abs(Y/params.width);
inside = (X_norm <= 0.5) & (Y_norm <= 0.5);
edge_dist = sqrt(min(0, 0.5 - X_norm).^2 + min(0, 0.5 - Y_norm).^2);
displacements(inside) = params.maxDisp * ...
(1 - 2*edge_dist(inside)).^2;
case 'tire'
R_norm = R/params.radius;
ring = R_norm >= 0.8 & R_norm <= 1.2;
displacements(ring) = params.maxDisp * ...
cos(pi * (R_norm(ring) - 1)).^2;
end
forces = k * displacements;
element_area = (X(1,2) - X(1,1)) * (Y(2,1) - Y(1,1));
pressures = forces / element_area;
end
function [vonMises, strainEnergy, pressureGrad] = calculateMetrics(X, Y,
displacements, pressures, params)
% Calculate strains
[dx, dy] = gradient(displacements, X(1,2)-X(1,1), Y(2,1)-Y(1,1));
% Calculate stresses
sigma_xx = params.E/(1-params.nu^2) * (dx + params.nu*dy);
sigma_yy = params.E/(1-params.nu^2) * (dy + params.nu*dx);
tau_xy = params.E/(2*(1+params.nu)) * (dx + dy);
% von Mises stress
vonMises = sqrt(sigma_xx.^2 + sigma_yy.^2 - sigma_xx.*sigma_yy + 3*tau_xy.^2);
% Strain energy density
strainEnergy = 0.5 * (sigma_xx.*dx + sigma_yy.*dy + tau_xy.*(dx+dy));
% Pressure gradient
[px, py] = gradient(pressures, X(1,2)-X(1,1), Y(2,1)-Y(1,1));
pressureGrad = sqrt(px.^2 + py.^2);
end
function plotResults(X, Y, displacements, forces, pressures, vonMises, strainEnergy,
pressureGrad, objectType, params)
figure('Position', [50 50 1500 900]);
```

```
% Plot displacement field
subplot(3,2,1)
surf(X*1000, Y*1000, -displacements*1000);
title(sprintf('Displacement Field - %s', objectType));
xlabel('X (mm)'); ylabel('Y (mm)'); zlabel('Displacement (mm)');
colorbar; view(45, 30); lighting gouraud; camlight;
% Plot pressure distribution
subplot(3,2,2)
surf(X*1000, Y*1000, -pressures/1e6);
title(sprintf('Pressure Distribution - %s', objectType));
xlabel('X (mm)'); ylabel('Y (mm)'); zlabel('Pressure (MPa)');
colorbar; view(45, 30); lighting gouraud; camlight;
% Plot von Mises stress
subplot(3,2,3)
surf(X*1000, Y*1000, -vonMises/1e6);
title('von Mises Stress');
xlabel('X (mm)'); ylabel('Y (mm)'); zlabel('Stress (MPa)');
colorbar; view(45, 30); lighting gouraud; camlight;
% Plot strain energy density
subplot(3,2,4)
surf(X*1000, Y*1000, -strainEnergy/1e3);
title('Strain Energy Density');
xlabel('X (mm)'); ylabel('Y (mm)'); zlabel('Energy Density (kJ/m³)');
colorbar; view(45, 30); lighting gouraud; camlight;
% Plot pressure gradient
subplot(3,2,5)
surf(X*1000, Y*1000, -pressureGrad/1e9);
title('Pressure Gradient');
xlabel('X (mm)'); ylabel('Y (mm)'); zlabel('Gradient (GPa/m)');
colorbar; view(45, 30); lighting gouraud; camlight;
% Plot contact area
subplot(3,2,6)
contact_threshold = max(pressures) * 0.05;
contact_area = pressures > contact_threshold;
contourf(X*1000, Y*1000, contact_area);
title('Contact Area');
xlabel('X (mm)'); ylabel('Y (mm)');
colormap(gca, [1 1 1; 0 0.4470 0.7410]);
colorbar('Ticks', [0.25, 0.75], 'TickLabels', {'No Contact', 'Contact'});
% Print analysis
fprintf('\nAnalysis Results for %s:\n', upper(objectType));
fprintf('Material Properties:\n');
fprintf('- Young''s Modulus: %.2e Pa\n', params.E);
fprintf('- Poisson''s Ratio: %.2f\n', params.nu);
fprintf('\nDeformation Metrics:\n');
fprintf('- Maximum Displacement: %.2f mm\n', max(displacements(:))*1000);
fprintf('- Maximum Pressure: %.2f MPa\n', max(pressures(:))/1e6);
```

```
fprintf('- Maximum von Mises Stress: %.2f MPa\n', max(vonMises(:))/1e6);
fprintf('- Maximum Strain Energy Density: %.2f kJ/m³\n', max(strainEnergy(:))/1e3);
fprintf('- Contact Area: %.2f mm²\n', sum(contact_area(:)) * (X(1,2)-X(1,1))*1000 *
(Y(2,1)-Y(1,1))*1000);
end
```

8.1 3DMesh.py

```
# First, install required packages
!pip install trimesh plotly pandas numpy

# Import required libraries
import trimesh
import numpy as np
import pandas as pd
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from google.colab import files

print("Please upload your XYZ point cloud file (Eraser_scan_1127_19_06_58.xyz)")
uploaded = files.upload()

print("\nNow please upload your OBJ mesh file (textured_output.obj)")
uploaded_mesh = files.upload()

# Read the original point cloud
points = pd.read_csv("Eraser_scan_1127_19_06_58.xyz", header=None, names=['x',
'y', 'z'])

# Load the mesh
mesh = trimesh.load("textured_output.obj")
vertices = np.array(mesh.vertices)
faces = np.array(mesh.faces)

# Rotate point cloud 90 degrees clockwise around X axis
rotation_matrix = np.array([
    [1, 0, 0],    # X-axis remains the same
    [0, 0, -1],   # Y-axis becomes -Z
    [0, 1, 0]     # Z-axis becomes Y
])

# Apply rotation to points
rotated_coords = points[['x', 'y', 'z']].values @ rotation_matrix.T
```

```
points['y'], points['z'] = rotated_coords[:, 1], rotated_coords[:, 2]

# Negate Y coordinates to match mesh orientation
points['y'] = -points['y']

# Generate deformation data
def generate_displacement(x, y, z):
    # Center coordinates of the depression
    x_center = 0.0
    y_center = 0.0

    # Calculate distance from center point
    r = np.sqrt((x - x_center)**2 + (y - y_center)**2)

    # Only apply displacement to points near the bottom (now in z-direction)
    if z < -0.02:
        # Gaussian depression with maximum 0.002 units displacement
        displacement = -0.002 * np.exp(-(r**2)/(2 * 0.02**2))
    else:
        displacement = 0.0

    return displacement

# Calculate displacements
displacements = points.apply(lambda row: generate_displacement(row['x'],
row['y'], row['z']), axis=1)

# Create deformed coordinates
points_deformed = points.copy()
points_deformed['z'] = points_deformed['z'] + displacements  # Apply to Z after
rotation

def create_comparison_visualization(orig_points, def_points, displacements,
mesh_vertices, mesh_faces):
    """
    Create side-by-side visualization of original and deformed states with mesh
    """
    # Create subplot figure with reduced spacing
    fig = make_subplots(
        rows=1, cols=2,
        specs=[[{'type': 'scene'}, {'type': 'scene'}]],
        subplot_titles=('Original State', 'Deformed State'),
        horizontal_spacing=0.02  # Reduce space between plots (default is 0.2)
```

```
    )

    # Add the mesh to both subplots
    for i in range(1, 3):
        fig.add_trace(
            go.Mesh3d(
                x=mesh_vertices[:, 0],
                y=mesh_vertices[:, 1],
                z=mesh_vertices[:, 2],
                i=mesh_faces[:, 0],
                j=mesh_faces[:, 1],
                k=mesh_faces[:, 2],
                color='lightblue',
                opacity=0.5,
                name="Original Mesh"
            ),
            row=1, col=i
        )

    # Add original point cloud to first subplot
    fig.add_trace(
        go.Scatter3d(
            x=orig_points['x'],
            y=orig_points['y'],
            z=orig_points['z'],
            mode='markers',
            marker=dict(
                size=2,
                color='red',
                opacity=0.8
            ),
            name="Original Points"
        ),
        row=1, col=1
    )

    # Add deformed point cloud to second subplot
    fig.add_trace(
        go.Scatter3d(
            x=def_points['x'],
            y=def_points['y'],
            z=def_points['z'],
            mode='markers',
```

```
            marker=dict(
                size=2,
                color=displacements,  # Color by displacement magnitude
                colorscale='Viridis',
                opacity=0.8,
                colorbar=dict(
                    title="Displacement (mm)",
                    x=1.0,  # Position colorbar on the right
                    len=0.8,  # Adjust colorbar length
                    thickness=15  # Make colorbar slightly thinner
                )
            ),
            name="Deformed Points"
        ),
        row=1, col=2
    )

    # Update layout with optimized spacing
    fig.update_layout(
        title="Comparison of Original and Deformed States",
        width=1800,
        height=800,
        showlegend=True,
        margin=dict(l=20, r=20, t=40, b=20)  # Reduce margins around the plots
    )

    # Set the same camera angle for both plots
    scene_settings = dict(
        xaxis_title="X",
        yaxis_title="Y",
        zaxis_title="Z",
        aspectmode='data',
        camera=dict(
            eye=dict(x=1.5, y=1.5, z=1.5)
        )
    )

    # Apply scene settings to both plots
    fig.update_scenes(scene_settings)

    return fig

# Create and display the visualization
```

```
fig = create_comparison_visualization(points, points_deformed, displacements,
vertices, faces)
fig.show()

# Save deformation data to CSV if needed
output = pd.DataFrame({
    'x_orig': points['x'],
    'y_orig': points['y'],
    'z_orig': points['z'],
    'x_def': points_deformed['x'],
    'y_def': points_deformed['y'],
    'z_def': points_deformed['z'],
    'displacement': displacements
})
output.to_csv('eraser_deformation.csv', index=False)
print("\nDeformation data saved to 'eraser_deformation.csv'")
```