# LUNAR SURFACE IMAGES STIMULATION AND VISUALISATION

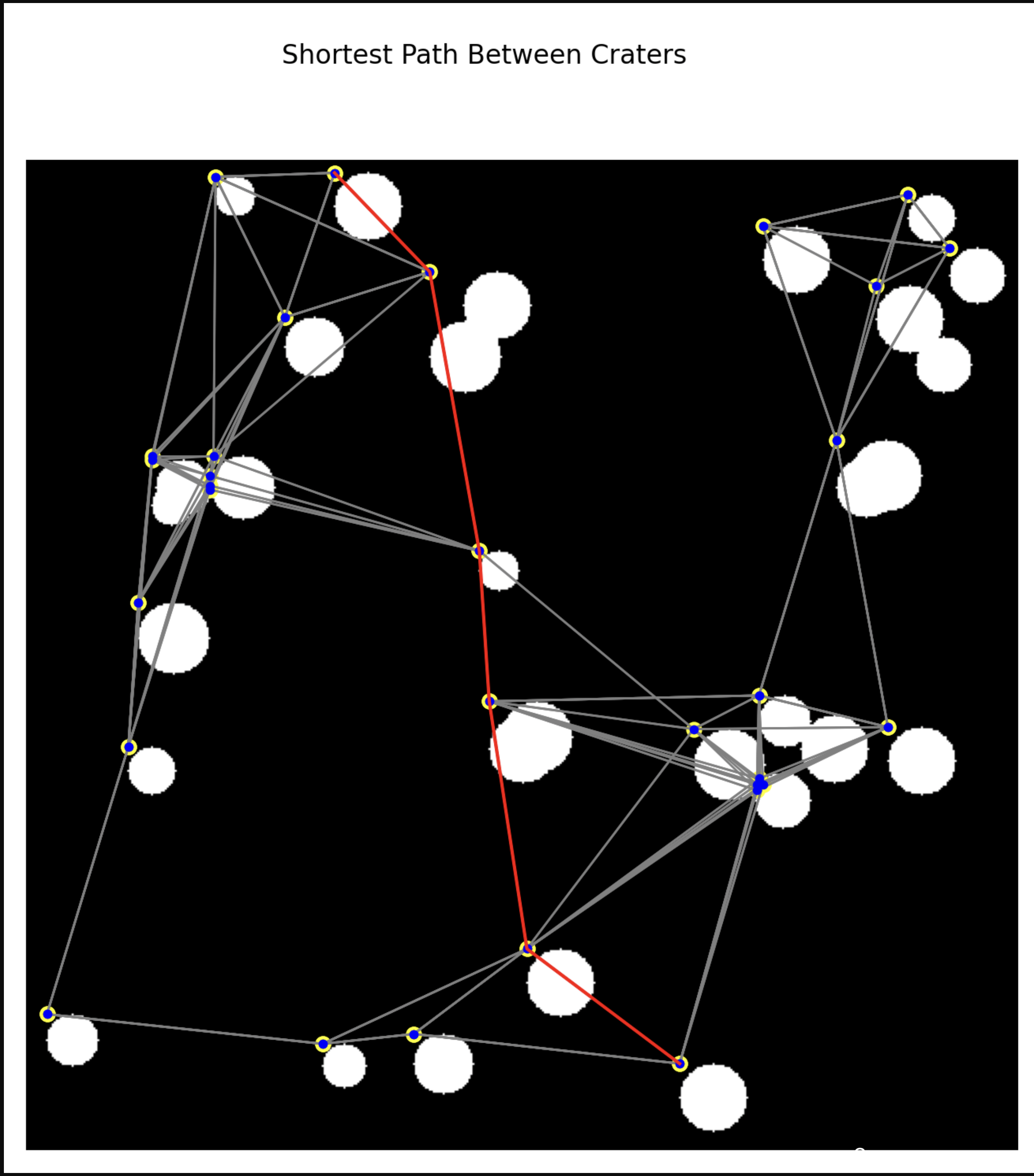
-Using Python for Crater Detection and Shortest Path Calculation-

7/24/2024

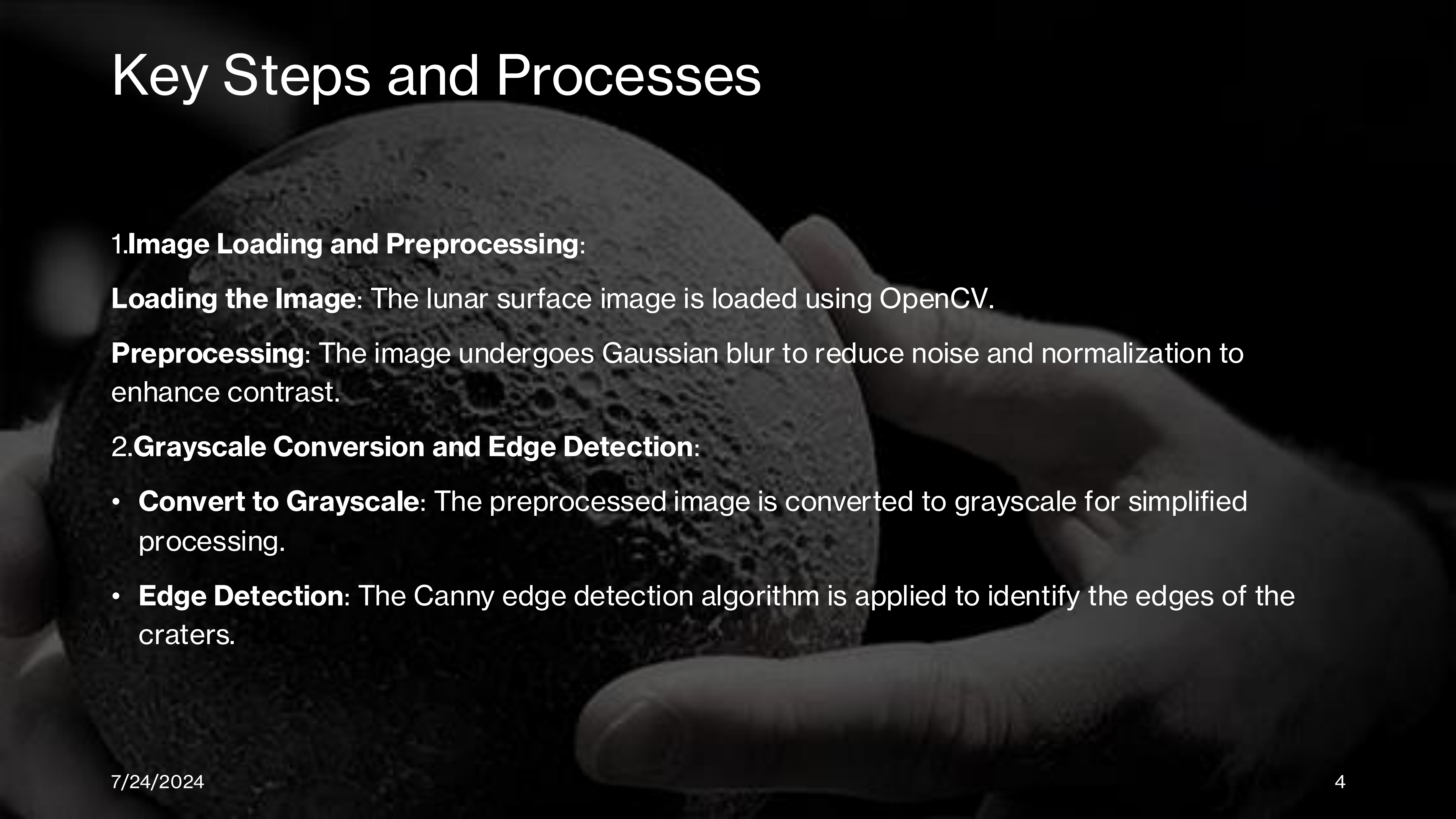
## Overview of Lunar Surface Image Processing and Navigation Path Visualization

In this project, we developed a Python script to process an image of the lunar surface captured by the Chandrayaan mission. The objective was to detect craters, extract relevant features, and determine the shortest navigation path between craters using graph theory

The final image showcases a synthetic lunar surface with white circles representing craters against a black background. Detected craters are highlighted with yellow dots to indicate their positions. The graph representation is overlaid, showing nodes as blue dots and edges as gray lines to depict the spatial relationships between craters. The shortest path between the start and end craters is traced with a red line, illustrating the optimal route. This visualization integrates crater detection and pathfinding, providing a clear view of crater distribution and the planned navigation route.



## Key Steps and Processes



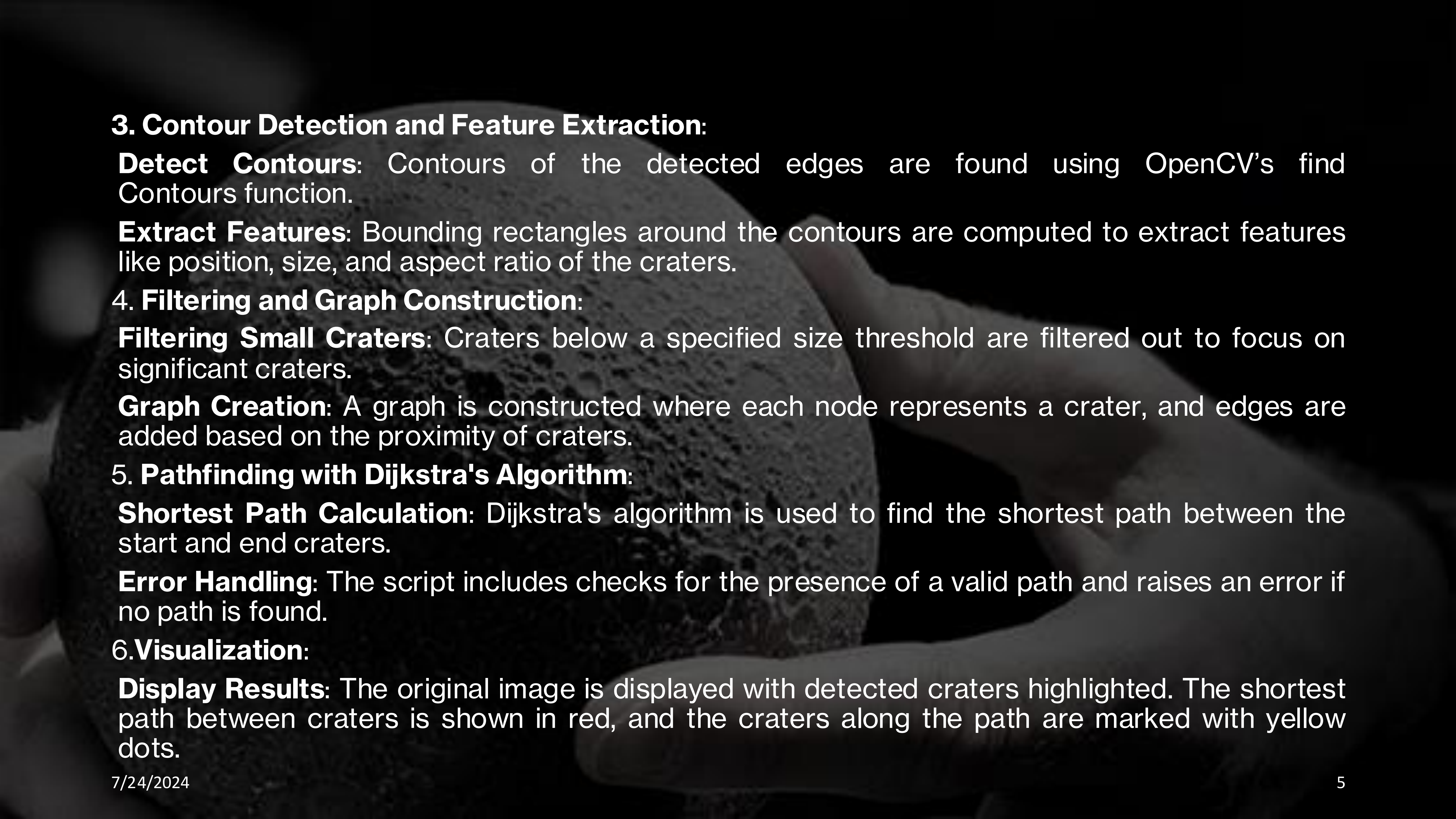
1.**Image Loading and Preprocessing**:

**Loading the Image**: The lunar surface image is loaded using OpenCV.

**Preprocessing**: The image undergoes Gaussian blur to reduce noise and normalization to enhance contrast.

2.**Grayscale Conversion and Edge Detection**:

- **Convert to Grayscale**: The preprocessed image is converted to grayscale for simplified processing.
- **Edge Detection**: The Canny edge detection algorithm is applied to identify the edges of the craters.



**3. Contour Detection and Feature Extraction**:

**Detect Contours**: Contours of the detected edges are found using OpenCV's find Contours function.

**Extract Features**: Bounding rectangles around the contours are computed to extract features like position, size, and aspect ratio of the craters.

4. **Filtering and Graph Construction**:

**Filtering Small Craters**: Craters below a specified size threshold are filtered out to focus on significant craters.

**Graph Creation**: A graph is constructed where each node represents a crater, and edges are added based on the proximity of craters.

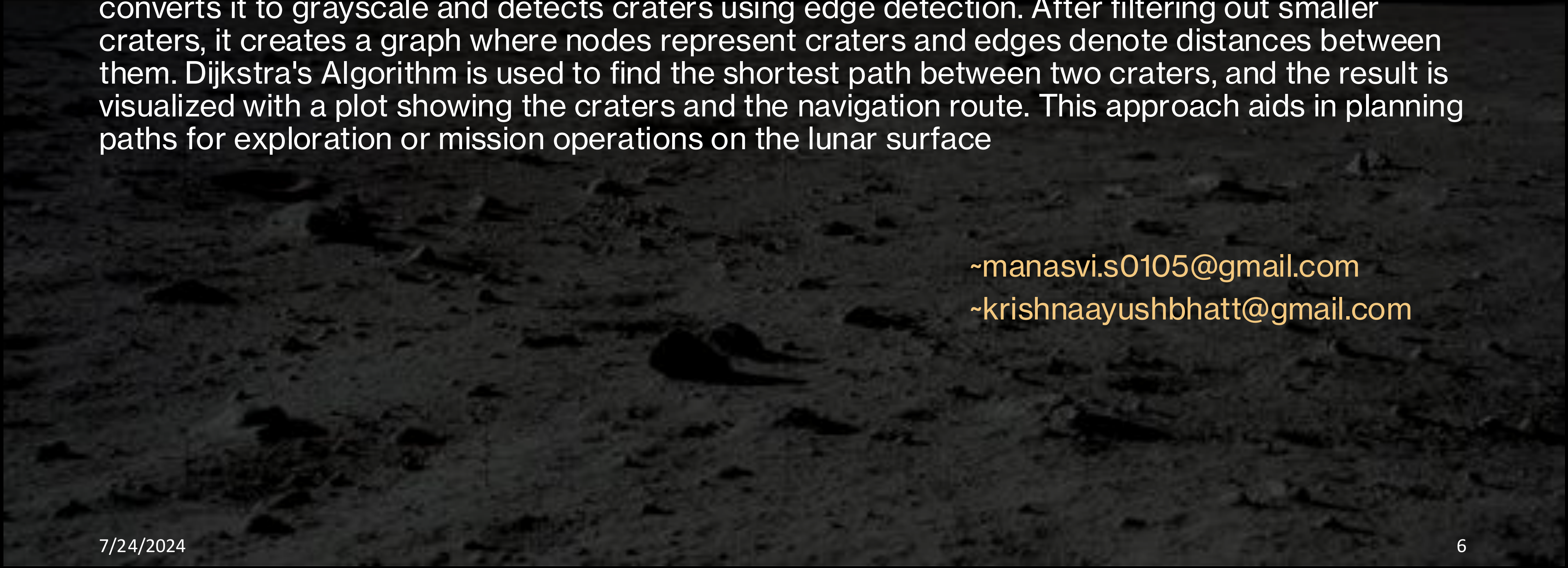
5. **Pathfinding with Dijkstra's Algorithm**:

**Shortest Path Calculation**: Dijkstra's algorithm is used to find the shortest path between the start and end craters.

**Error Handling**: The script includes checks for the presence of a valid path and raises an error if no path is found.

6.**Visualization**:

**Display Results**: The original image is displayed with detected craters highlighted. The shortest path between craters is shown in red, and the craters along the path are marked with yellow dots.



The code processes a lunar surface image to detect craters and determine a navigational path between them. It begins by preprocessing the image with Gaussian blur and normalization, then converts it to grayscale and detects craters using edge detection. After filtering out smaller craters, it creates a graph where nodes represent craters and edges denote distances between them. Dijkstra's Algorithm is used to find the shortest path between two craters, and the result is visualized with a plot showing the craters and the navigation route. This approach aids in planning paths for exploration or mission operations on the lunar surface

~manasvi.s0105@gmail.com

~krishnaayushbhatt@gmail.com