DIGITAL IMAGE PROCESSING
TERM PROJECT

AN OBSTACLE DETECTION METHOD FOR PLANETARY ROVERS

Ayşe Atik
23050951014

04/01/2026

Abstract

Planetary rovers operate in unknown and dangerous environments that require reliable obstacle detection for safe navigation and mission continuity. Stereo vision is commonly used in planetary exploration because it provides depth information with lightweight, low-power camera systems. However, images of planetary surfaces are often affected by challenging conditions such as changes in illumination, shadows, and rough terrain. In addition, 2D grayscale images do not contain depth information, while 3D data obtained from stereo vision is often sparse or incomplete. For these reasons, obstacle detection methods that rely solely on 2D or 3D information may be less reliable. In this study, a hybrid obstacle detection approach that combines 2D grayscale image information with sparse 3D point cloud data was examined and implemented in a similar way. The goal is to understand the working principles of this method and to evaluate its effectiveness under different conditions through experimental analysis through simulations.

Contents

1. Introduction

1.1 Background and Motivation

Planetary rovers navigate unknown and dangerous terrains. Obstacles like rocks, slopes, and craters can terminate missions. Detecting these obstacles is crucial for navigation safety. While 2D images are lightweight and easy to capture, they cannot provide depth information. On the other hand, stereo vision is capable of giving 3D data but may still be sparse or unreliable under poor lighting and uneven terrain.

1.2 Problem Definition

The challenge lies in detecting obstacles reliably despite the limitations of both 2D and 3D data. Sparse stereo measurements and incomplete depth information make it difficult to fully assess terrain geometry. Combining 2D image features with 3D stereo data can improve obstacle detection accuracy.

1.3 Objectives of the Study

This project implements a hybrid obstacle detection method inspired by *“Vision-Based Obstacle Detection Using Rover Stereo Images”* [1]. The reference combines 2D grayscale images with sparse 3D point information to design a more reliable obstacle detection method for planetary rovers. In our study, we aim to understand and implement a simplified adaptation of this method, gain hands-on experience, and evaluate its performance in a simulated planetary environment. Unlike the original work, our approach uses dense stereo matching to generate 3D point clouds and stereo images obtained from a Unity simulation rather than real planetary data. Additional simplifications in our method are discussed later.

2. Literature Review

2.1 Overview of Obstacle Detection

Stereo vision is commonly used in planetary rover missions to detect obstacles, as it provides depth information with lightweight, low-power cameras [1]. By capturing images from slightly different viewpoints, stereo cameras enable 3D reconstruction through triangulation and feature matching. However, environmental factors such as shadows, illumination variations, and uneven terrain often produce sparse or incomplete 3D point clouds, limiting the reliability of methods based solely on 3D data [1], [2].

2.2 Related Works

Obstacle detection approaches can exploit 2D image data, 3D stereo information, or a combination of both. 2D methods include edge-based, region-based, and machine learning techniques, such as Canny edge detection [3], K-means clustering for region segmentation [4], and cascade classifiers [5]. These approaches can detect objects like rocks but often lack precise boundary information.

3D-based methods reconstruct point clouds from stereo images to identify terrain features and obstacles [2], [6], [7]. Although they provide distance information, fine-grained texture and grayscale details are usually lost. Hybrid approaches that integrate 2D and 3D data have been proposed to address these limitations. For example, Wang et al. [1] combined grayscale images with sparse 3D point clouds from rover stereo cameras using mean-shift segmentation and statistical metrics, demonstrating improved obstacle detection under challenging conditions.

For this project, the method of Wang et al. [1] was selected because it focuses specifically on stereo-image-based obstacle detection, which aligns closely with the course's learning objectives. The method is suitable for the study because it relies primarily on images captured from stereo cameras rather than complex sensor arrays, making it accessible for simulation and educational purposes. Implementing this method in a simplified way allows hands-on experience in 2D-3D data fusion, parameter tuning, and evaluating detection performance in a simulated planetary environment.

3. Dataset Description

3.1 Dataset Source

For this project, a custom Mars environment with some stone obstacles was created in Unity to simulate planetary terrain. Stereo image pairs were captured directly from this environment using two virtual cameras placed at a fixed baseline (0.06m) on a cube with a degree of 46, mimicking a rover's stereo setup [8]. This approach allowed us to control lighting, terrain, and object placement while providing a testbed for obstacle detection algorithms.

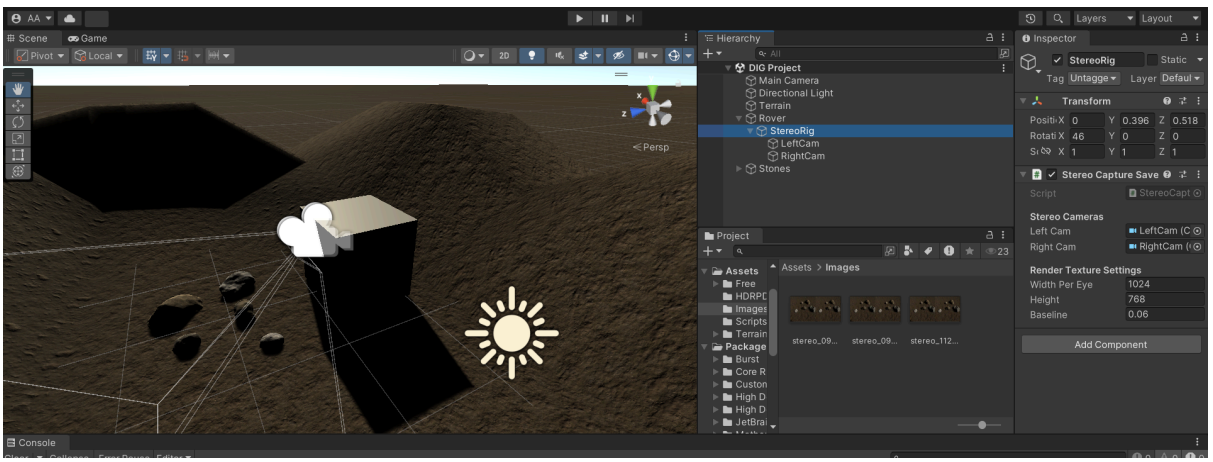

Image 1: The Unity environment for capturing required stereo images

3.2 Dataset Characteristics

The captured stereo images are colored and have a resolution of 1024×768 pixels. The simulated scene contains only the terrain and rocks of various sizes, without any additional objects, and the images are free from motion blur or noise due to the

controlled environment. Each stereo pair represents a single viewpoint of the terrain, with the left image used for segmentation and visualization.

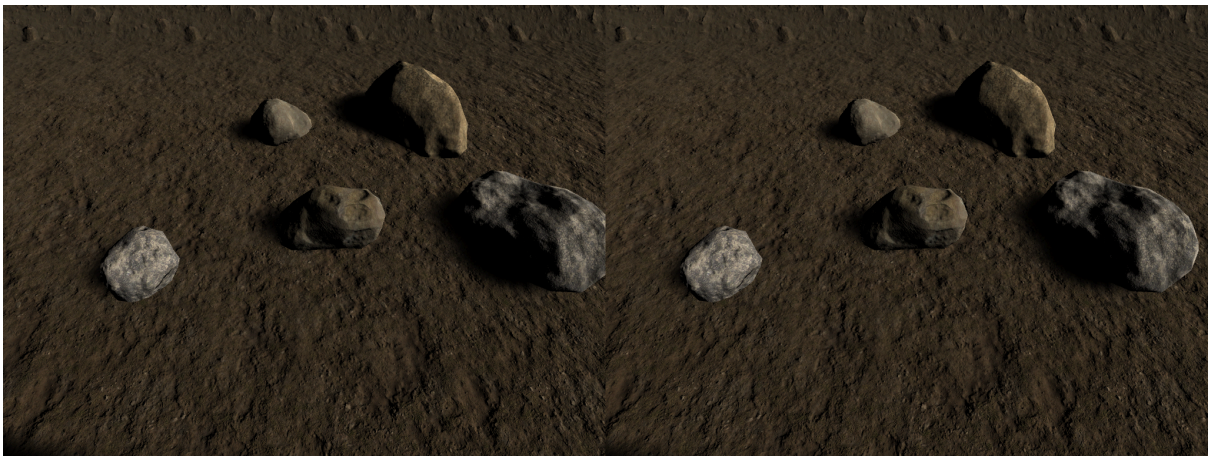

Image 2: A sample captured stereo image from the scene

3.3 Preprocessing Steps

No additional preprocessing was required for the images, as the simulation environment provided clean, well-illuminated images. The images were simply converted to grayscale before processing. Dense stereo matching was applied directly to compute disparity maps. Although the simulation provides ideal conditions and lacks real-world noise or illumination variations, optional enhancements, such as histogram equalization, could be applied to improve contrast and further support stereo matching in more realistic scenarios.

4. Methodology

4.1 Proposed Method

Our study is inspired by the hybrid obstacle detection method proposed by Wang et al. [1], which combines 2D grayscale images with 3D stereo data to improve obstacle detection reliability. The original method [1], detects obstacles by combining 2D grayscale image information with 3D point clouds obtained from rover stereo images. Sparse feature points are first extracted from rectified stereo pairs using the Förstner operator and matched using correlation and RANSAC filtering. Dense matching is then guided by these matched points to reconstruct 3D terrain with sub-pixel accuracy. Meanwhile, the left image is segmented using the mean-shift algorithm to identify candidate regions based on spatial and intensity information. 3D points are projected onto the image, and regions containing a sufficient number of points are considered seed regions. For each seed region, terrain roughness and elevation step metrics are calculated, while non-seed regions are assessed based on grayscale intensity. Regions exceeding predefined thresholds are classified as obstacles, and morphological post-processing is applied to generate the final obstacle map [1].

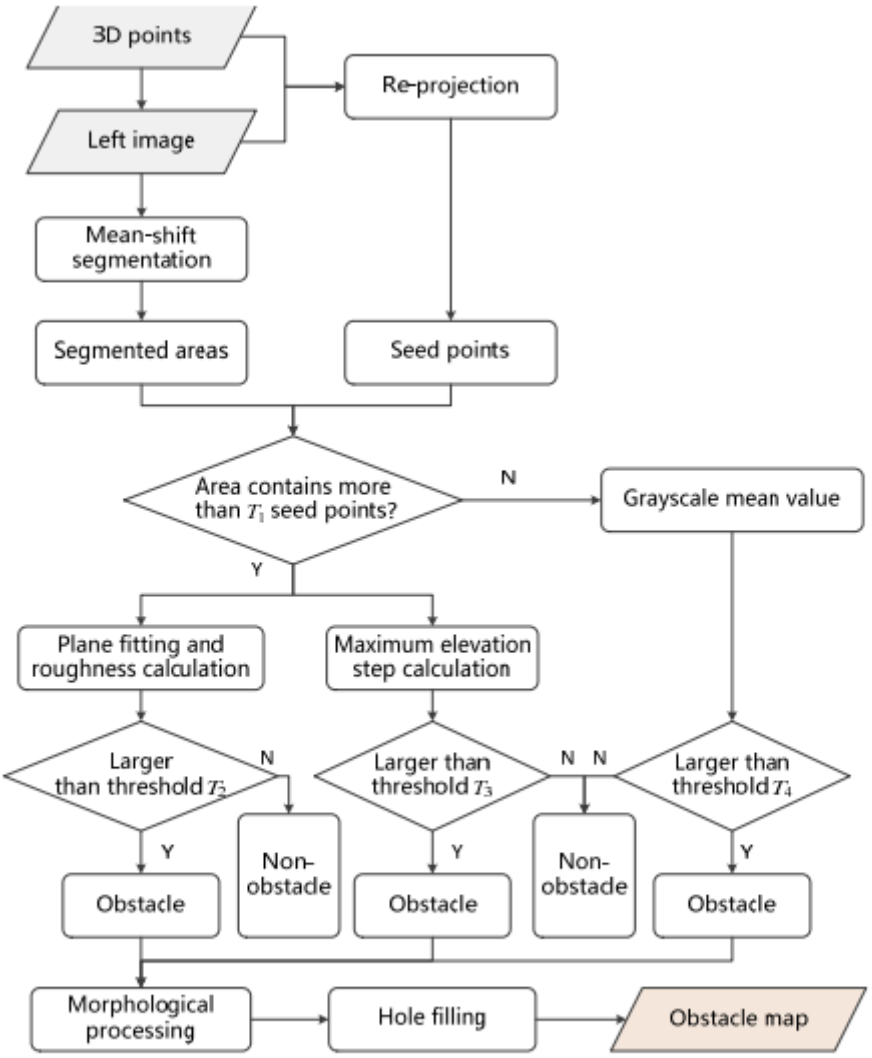

Figure 1. Flowchart of the obstacle detection strategy combining 2D and 3D information

Image 3: The flowchart of the method by Wang et al. [1]

Our method generates a dense 3D point cloud directly from stereo image pairs using dense stereo matching, unlike the original approach which relies on sparse feature points [1]. The left image is converted to grayscale and segmented with the mean-shift algorithm to identify candidate regions. Seed points are determined by projecting the dense 3D points onto these segmented regions, rather than using sparse projections. Within each region, terrain roughness and elevation step metrics are calculated from the 3D points, and regions exceeding predefined thresholds are classified as obstacles. For regions with insufficient 3D data, the grayscale mean value is used to detect potential obstacles. Finally, morphological operations are applied to refine the obstacle map, producing a complete and smooth representation of obstacles without additional preprocessing or external filtering. These adaptations simplify the original method.

4.2 Implementation Details and Algorithms

The step-by-step implementation of our method is as follows:

1. **Image Acquisition:** Stereo images were captured in a Unity-based Mars-like environment with a resolution of 1024×768 pixels. The images are colored, with a camera tilt of 46° and a baseline of 0.06 m. They were then converted to grayscale before processing, providing a convenient input for segmentation and dense stereo matching.

2. **Dense Stereo Matching**: Disparity maps are computed from the left and right stereo images using OpenCV's StereoSGBM_create() algorithm. This method performs block-based semi-global matching, where small windows of pixels (blockSize = 5) are compared across the images to calculate pixel-wise disparities. Parameters such as numDisparities, minDisparity, P1, P2, uniquenessRatio, and speckleWindowSize control the disparity search range, smoothness penalties, and filtering of mismatches. The resulting disparity map encodes relative depth at each pixel, and median filtering is applied to reduce noise while preserving edges. This produces a dense disparity map that is used for 3D reprojection.

```
# --- 1) DENSE STEREO MATCHING ---
stereo = cv2.StereoSGBM_create(
    minDisparity=0,         # Minimum disparity value to be considered.
    numDisparities=16*3,    # Range of disparity values to search.
    blockSize=5,            # Size of the block/window used for matching.
    P1=8*3*5*5,             # Penalty on the disparity change by ±1 between neighboring pixels.
    P2=32*3*5*5,            # Penalty on the disparity change by more than 1 between neighboring pixels.
    disp12MaxDiff=1,        # Maximum allowed difference in the left-right disparity check.
    uniquenessRatio=8,      # Margin in percent by which the best match should be better than the second-best.
    speckleWindowSize=50,   # Maximum size of smooth disparity regions to consider as speckle noise.
    speckleRange=2,         # Maximum disparity variation within each connected component to detect speckles.
    preFilterCap=63,        # Truncation value for prefiltered image pixels.
    mode=cv2.STEREO_SGBM_MODE_SGBM_3WAY  # Use 3-way semi-global matching for better accuracy.
)
disparity = stereo.compute(left_img, right_img).astype(np.float32)/16.0
disparity[disparity<=0] = 0
disparity = cv2.medianBlur(disparity,5)

os.makedirs("debug_outputs", exist_ok=True)
disp_vis = cv2.normalize(disparity, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
cv2.imwrite("debug_outputs/disparity.png", disp_vis)
```

3. **3D Reprojection**: Using the computed disparity map from dense stereo matching, each pixel is reprojected into 3D space to obtain a point cloud. First, the camera intrinsic parameters (focal lengths fx, fy and principal point cx, cy) and the baseline distance between the stereo cameras are used to construct the Q matrix. This matrix enables mapping from disparity values to real-world 3D coordinates. The cv2.reprojectImageTo3D() function applies the Q matrix to the disparity map to calculate the X, Y, Z coordinates for each pixel. Points with invalid or out-of-range depths (less than 0 m or greater than 10 m) are masked out. The resulting point cloud is dense, containing 3D coordinates for nearly every pixel in the valid regions of the image, representing the terrain geometry for subsequent obstacle analysis.

```
# --- 2) 3D RE-PROJECTION ---
h, w = left_img.shape
fov_vertical_deg = 60
aspect = w/h
fov_rad = math.radians(fov_vertical_deg)
fy = h/(2*math.tan(fov_rad/2))
fx = fy*aspect
cx, cy = w/2, h/2
baseline = 0.06
Q = np.float32([[1,0,0,-cx],[0,1,0,-cy],[0,0,0,fx],[0,0,1/baseline,0]])
points_3d = cv2.reprojectImageTo3D(disparity, Q)
Z = points_3d[:,:,2]
mask_3d = (Z>0) & (Z<10)
points = points_3d[mask_3d]
colors = cv2.cvtColor(left_color, cv2.COLOR_BGR2RGB)[mask_3d]/255.0
```

4. **Mean-Shift Image Segmentation**: The left grayscale image is first converted to BGR and filtered using the mean-shift algorithm with a spatial bandwidth of sp = 15 and a color bandwidth of sr = 20. Thresholding and connected components labeling are then applied to remove small regions (<50 pixels). This process segments the image into homogeneous regions based on spatial and grayscale similarity, identifying candidate regions for obstacle evaluation.

```
# --- 3) MEAN-SHIFT SEGMENTATION ---
left_bgr = cv2.cvtColor(left_img, cv2.COLOR_GRAY2BGR)
mean_shifted = cv2.pyrMeanShiftFiltering(left_bgr, sp=15, sr=20)
segmented_gray = cv2.cvtColor(mean_shifted, cv2.COLOR_BGR2GRAY)
_, segmented_bin = cv2.threshold(segmented_gray,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
num_labels, labels = cv2.connectedComponents(segmented_bin)
min_region_size = 50
cleaned_labels = np.zeros_like(labels)
current_label = 1
for label in range(1,num_labels):
    region_mask = (labels==label)
    if np.sum(region_mask)>=min_region_size:
        cleaned_labels[region_mask] = current_label
        current_label+=1
```

5. **Seed Points Determination**: 3D points are projected back onto the segmented image. For each segmented region, the number of projected 3D points is counted, and regions containing 10 or more points are marked as seed regions. These seed regions indicate areas with sufficient depth information for further obstacle analysis.

```
# --- 4) SEED POINTS ---
T1 = 10
h, w = cleaned_labels.shape
num_regions = np.max(cleaned_labels)
region_seed_count = np.zeros(num_regions+1, dtype=int)
valid_mask = (Z>0) & (Z<10)
ys, xs = np.where(valid_mask)
for y,x in zip(ys,xs):
    label = cleaned_labels[y,x]
    if label>0:
        region_seed_count[label]+=1
seed_region_mask = np.zeros_like(cleaned_labels, dtype=bool)
for label in range(1,num_regions+1):
    if region_seed_count[label]>=T1:
        seed_region_mask[cleaned_labels==label]=True
cv2.imwrite("debug_outputs/seed_regions.png",
            (seed_region_mask.astype(np.uint8)*255))
```

6. **Obstacle Evaluation**: For each seed region, a sliding window of 5×5 pixels is applied. Within each window, the 3D points are used to fit a plane using least-squares, from which terrain roughness (average deviation from the plane) and maximum elevation step (difference between highest and lowest points) are calculated. Regions exceeding the thresholds T2 = 0.0005 for roughness or T3 = 0.005 for elevation step are classified as obstacles. Seed regions with insufficient 3D points are instead evaluated using the mean grayscale intensity, and regions with intensity below T4 = 50 are also considered obstacles.

```
# --- 5) OBSTACLE DETECTION ---
window_size = 5
T2 = 0.0005
T3 = 0.005
T4 = 50
obstacle_map = np.zeros_like(cleaned_labels, dtype=bool)
rough_step_map = np.zeros_like(cleaned_labels, dtype=np.uint8)

for region_label in range(1,num_regions+1):
    region_mask = (cleaned_labels==region_label)
    ys, xs = np.where(region_mask)
    if len(ys)<10: continue
    y_min, y_max = ys.min(), ys.max()
    x_min, x_max = xs.min(), xs.max()
    for y_start in range(y_min, y_max, window_size):
        for x_start in range(x_min, x_max, window_size):
            y_end = min(y_start+window_size, y_max+1)
            x_end = min(x_start+window_size, x_max+1)
            window_mask = region_mask[y_start:y_end, x_start:x_end]
            if np.sum(window_mask)<3: continue
            window_points = points_3d[y_start:y_end, x_start:x_end,:]
            window_points = window_points[window_mask]
            window_points = window_points[np.isfinite(window_points).all(axis=1)]
            if len(window_points)<3: continue
            X = window_points[:,:2]
            Z_vals = window_points[:,2].reshape(-1,1)
            A = np.hstack([X,np.ones((X.shape[0],1))])
            try:
                coeff, _, _, _ = np.linalg.lstsq(A,Z_vals,rcond=None)
                plane_z = A@coeff
                roughness = np.mean(np.abs(Z_vals-plane_z))
                step = np.max(Z_vals)-np.min(Z_vals)
                rough_step_map[y_start:y_end, x_start:x_end] = int(min(255,(roughness+step)*100))
                if roughness>T2 or step>T3:
                    mask_to_update = np.zeros_like(region_mask,dtype=bool)
                    mask_to_update[y_start:y_end, x_start:x_end] = True
                    obstacle_map |= (mask_to_update & region_mask)
```

```
            except np.linalg.LinAlgError:
                continue
    if region_seed_count[region_label]<T1:
        mean_gray = np.mean(left_img[region_mask])
        if mean_gray<T4:
            obstacle_map |= region_mask

cv2.imwrite("debug_outputs/rough_step_map.png", rough_step_map)
```

7. **Morphological Post-Processing**: The detected obstacle map undergoes morphological closing and opening using a 7×7 elliptical kernel with two iterations each. This step smooths region contours, fills small holes, and generates continuous obstacle regions suitable for navigation.

```
# --- 6) MORPHOLOGICAL POST-PROCESSING ---
obstacle_map_img = obstacle_map.astype(np.uint8)*255
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(7,7))
obstacle_map_img = cv2.morphologyEx(obstacle_map_img, cv2.MORPH_CLOSE, kernel, iterations=2)
obstacle_map_img = cv2.morphologyEx(obstacle_map_img, cv2.MORPH_OPEN, kernel, iterations=2)
cv2.imwrite("debug_outputs/final_obstacle_map.png", obstacle_map_img)
print(f"Detected obstacles in {int(np.sum(obstacle_map))} pixels.")

return jsonify({"status":"success","num_obstacles":int(np.sum(obstacle_map))})
```

This step-by-step implementation mirrors the hybrid approach of Wang et al. [1], while adapting it to our simulated Unity environment. It allows testing and evaluating obstacle detection under controlled yet realistic conditions.

5. Experimental Results

5.1. Evaluation Metrics

In this project, the algorithm parameters (e.g., window size, thresholds T1–T4, mean-shift bandwidths) were empirically optimized during the implementation phase through visual inspection of the outputs. The quantitative evaluation of the obstacle detection method will be conducted later using metrics such as pixel-wise detection accuracy, precision, and recall, which measure how accurately obstacles are identified and the occurrence of false positives or false negatives. Additionally, terrain roughness and elevation step distributions can be analyzed to verify the effectiveness of the chosen thresholds in distinguishing obstacles from traversable areas. The method will also be tested on different scenes by changing the simulated environment to ensure its robustness across varying terrain conditions.

5.2 Results and Visual Outputs

The outputs from the implementation - including disparity maps, segmented regions, seed points, roughness/elevation maps, and the final obstacle map - are currently used to visually validate the method and demonstrate its effectiveness on the simulated terrain. The input image was image 2 in the Data Description section.

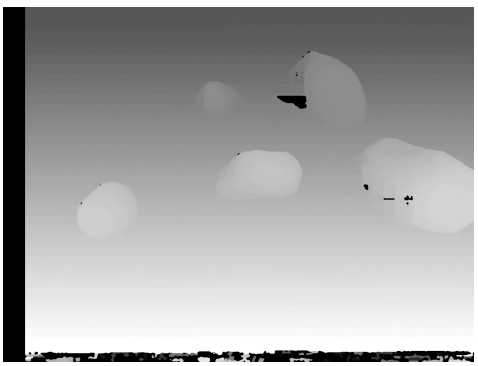

Image 5: Disparity map

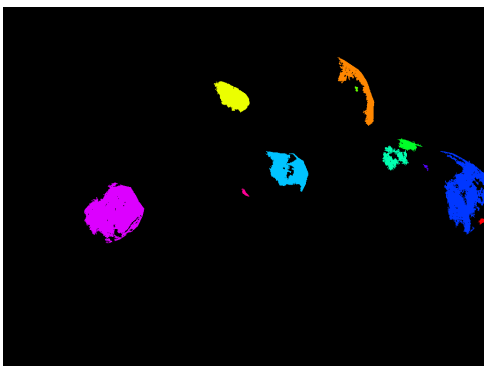

Image 6: Segmented regions

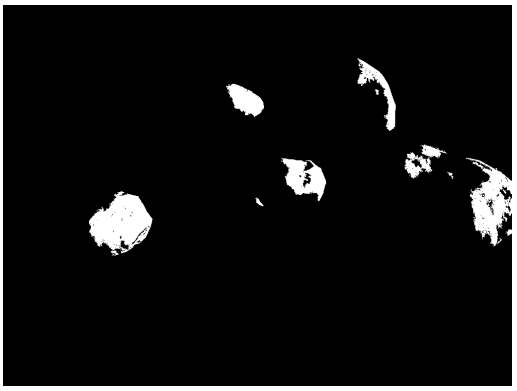

Image 7: Seed regions

Image 8: Rough-step map

Image 9: Final obstacle map

6. Conclusions

6.1 Summary

This project presented a hybrid obstacle detection method that combines 2D grayscale image information with dense 3D point clouds obtained via stereo

matching. The implementation included mean-shift segmentation, seed point determination, terrain roughness and elevation step analysis, and morphological post-processing to produce a final obstacle map. The method was tested on a simulated Mars-like environment, demonstrating its capability to identify obstacles such as rocks, slopes, and depressions. The pipeline allows step-by-step visualization of disparity maps, segmented regions, seed points, and roughness/elevation maps, providing a clear understanding of the detection process.

6.2 Limitations and Future Work

The current implementation relies on a simulated environment, which does not capture all complexities of real planetary terrain, such as varying illumination, shadows, or sensor noise. The evaluation metrics for quantitative assessment are planned but not yet applied. Future work includes testing the method on multiple diverse scenes, integrating real stereo camera data, and exploring adaptive or learning-based thresholding to improve robustness under different environmental conditions.

References

[1] Wang, X., et al., “Vision-Based Obstacle Detection Using Rover Stereo Images,” *Journal of Field Robotics*, vol. 36, no. 5, pp. 857–874, 2019.

[2] Matthies, L., et al., “Obstacle Detection for Planetary Rovers: Stereo Vision Approaches,” *Robotics and Autonomous Systems*, vol. 54, no. 2, pp. 163–171, 2008.

[3] Castano, A., et al., “Edge Detection Techniques for Planetary Rover Applications,” *IEEE Trans. Robotics*, vol. 21, no. 4, pp. 789–798, 2005.

[4] Bajracharya, M., “Region-Based Obstacle Detection for Mobile Robots,” *Proc. IEEE Int. Conf. Robotics and Automation*, pp. 123–130, 2002.

[5] Viola, P., and Jones, M., “Rapid Object Detection Using a Boosted Cascade of Simple Features,” *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp. 511–518, 2001.

[6] Lagisetty, R., et al., “3D Terrain Reconstruction from Rover Stereo Images,” *Int. J. Robotics Research*, vol. 32, no. 6, pp. 689–705, 2013.

[7] Liu, H., et al., “Stereo-Based Obstacle Detection and Mapping for Planetary Rovers,” *Robotics*, vol. 4, no. 3, pp. 234–247, 2015.

[8] **“**Create side-by-side stereo pairs in the Unity game engine”, online: http://www.paulbourke.net/stereoscopy/Unitystereo/