Obstacle Detection Planning (LRM)

Objective

Figure out an efficient way to implement obstacle detection on Lunar Rover using Sentis-ToF-M100 LiDAR sensor.

Obstacle Detection Algorithm

High Level Overview

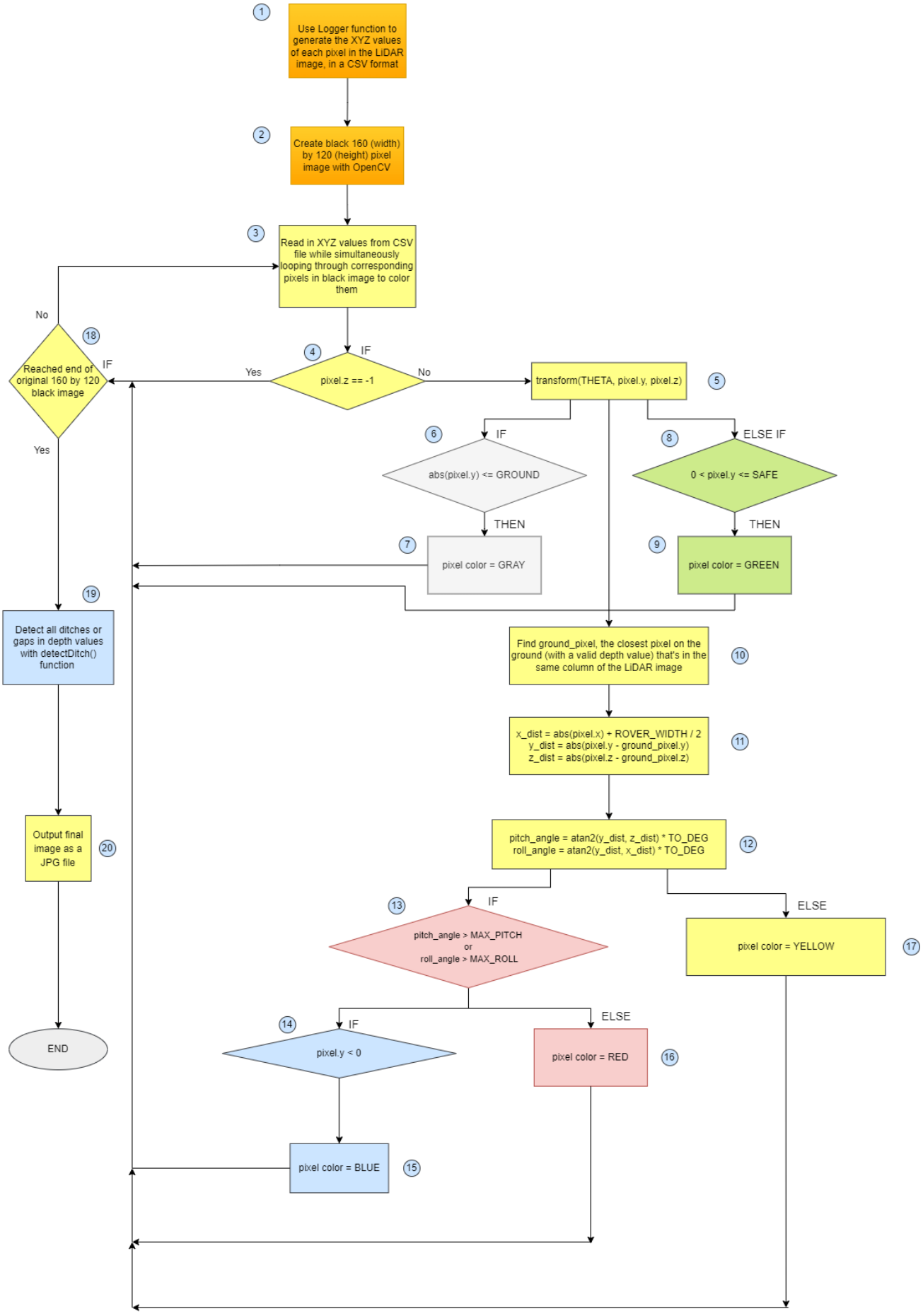
My algorithm is designed under the assumption that the LiDAR is mounted at the centre, highest point on the front face of the Rover. I plan on using a slope-based obstacle detection algorithm that first transforms the XYZ coordinates of each detected pixel in a 160 by 120 LiDAR image generated by the ToF sensor, from the ground's reference frame to the LiDAR's reference frame, before calculating the pixel's slope with reference to a detected pixel that is closest to the ground and in the same column of the LiDAR image.

The algorithm starts by creating a black 160 by 120 pixel image. Then, the XYZ values of each pixel on the 160 by 120 pixel LiDAR image are read in while simultaneously iterating through the corresponding pixels in the black image to color these black pixels based on a few conditions.

If the pixel has a valid depth value detected by the LiDAR, the pixel is transformed from the ground's reference frame to the LiDAR's reference frame. The pixel's slope with reference to the closest detected pixel on the ground (in the same column), is used to calculate its pitch and roll Euler angles. Based on these angles, the pixel will then be assigned a specific color (RED = Large Pitch or Large Roll, YELLOW = Warning Zone, GREEN = Safe Zone, and BLUE = Ditch) where red represents a large slope that should not be driven over, yellow represents an area or slope that should be driven over with caution, and green represents an area or obstacle that can be safely traversed without worry. If a pixel below the ground has a large pitch or roll angle, it will be colored navy blue to represent a ditch.

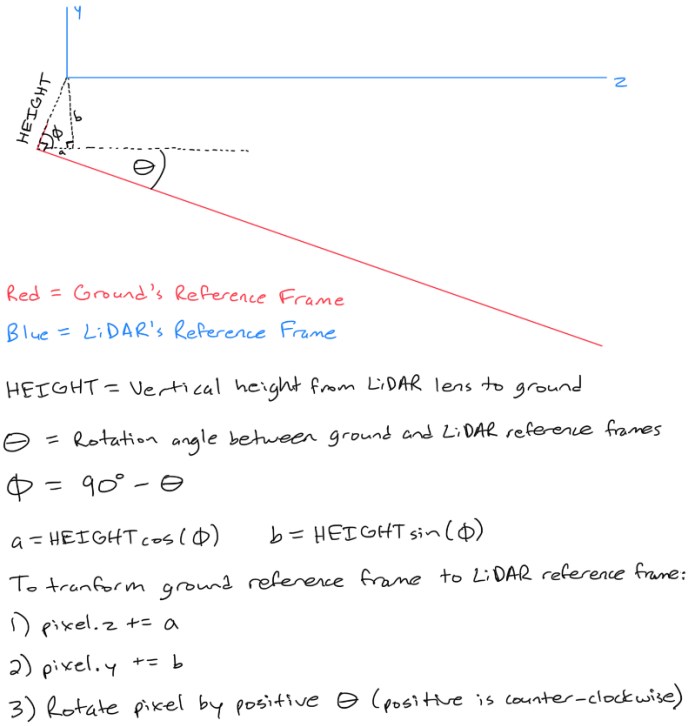
If the pixel's absolute value height from the ground is below 100 mm, it will be colored gray to represent the ground or resemble an occupancy grid indicating obstacle-free zones the operator could drive the Rover towards. The pixels in the LiDAR image that have no depth value, will remain black in the 160 by 120 pixel black image. The detectDitch() function will then color all pixels that could represent ditches or gaps in depth value between two objects in the LiDAR image blue. Finally, the final image will be outputted to a JPG file.

Here is a flowchart representation of the algorithm:



Here is a step-by-step explanation of each block of the algorithm:

1. Use the Logger function in the Sentis ToF API to generate the XYZ values of each pixel in the LiDAR image, in a CSV format.
2. Use OpenCV to create a black 160 (width) by 120 (height) pixel image.
3. Read in the CSV files to read in the XYZ values of each pixel in the LiDAR image, while simultaneously iterating through the corresponding pixels in the 160 by 120 pixel black image to color these corresponding pixels.
4. Check if the pixel from the LiDAR image has a valid depth value (z value of -1 means invalid depth value).
5. Since from the LiDAR's point-of-view the ground is tilted upwards by a positive angle even though the ground is completely flat in real life, this means that the LiDAR's reference frame is tilted upwards (meaning every point in the LiDAR's point-of-view is also tilted upwards) from the ground's reference frame by this angle. If the pixel has a valid depth value (z value != -1), we plug in THETA (the angle by which the ground's reference frame will be rotated about the x-axis to become parallel with the LiDAR's reference frame), the pixel's y value, and the pixel's z value into the transform() function which transforms the pixel from the ground's reference frame to the LiDAR's reference frame. See drawing below for explanation. The angle THETA was determined using the Linear Regression algorithm described later in this document.



6. After the transformation, if the pixel's absolute value height from the ground is below or equal to 100 mm (GROUND), the pixel can be considered a part of the ground.
7. Since the pixel can be considered ground, it will be colored gray. Continue to the next iteration of the algorithm.
8. If the pixel's height from the ground is positive and less than or equal to 150 mm (SAFE), the pixel can be considered "safely traversable without worry".

9. Since the pixel can be considered “safely traversable without worry”, it will be colored green. Continue to the next iteration of the algorithm.
10. Use a while loop to find the closest pixel on the ground that’s in the same column of the LiDAR image and save its XYZ values in another Pixel object called ground_pixel.
11. Store the magnitudes of the y and z distances from ground_pixel to the pixel, in the y_dist and z_dist variables. Since the centre of the LiDAR also represents the centre point between the front wheels of the Rover, add half the width of the Rover to the x distance to store the distance from the pixel to the farthest point on the front wheel that the Rover will pivot on when rolling. Store this distance in the variable x_dist. The slope of the pixel with reference to the front wheel that the Rover will pivot on when rolling, will be used to calculate the Roll angle.
12. Take the inverse tan of the y_dist divided by the z_dist and convert to degrees to determine the pitch angle. Take the inverse tan of y_dist divided by x_dist and convert to degrees to determine the roll angle.
13. If the pitch angle is greater than the Max Pitch threshold or the roll angle is greater than the Max Roll threshold, the pixel will be considered “dangerous” indicating that the Rover should not attempt to drive over it.
14. If the pixel exceeds the Max Pitch or Max Roll thresholds, check if it’s below the ground.
15. If the pixel exceeds the Max Pitch or Max Roll thresholds and it’s below the ground, color it blue as it represents a ditch. Continue to the next iteration of the algorithm.
16. If the pixel exceeds the Max Pitch or Max Roll thresholds and it’s above the ground, color it red as it represents a large slope that cannot be traversed. Continue to the next iteration of the algorithm.
17. Else, in the case that the pixel doesn’t exceed the Max Pitch or Max Roll thresholds, it will be considered “safe but proceed with caution” and will be colored yellow. Continue to the next iteration of the algorithm.
18. Check if the algorithm has reached the end of the original 160 by 120 black image and continue iterating through the rest of the pixels in the 160 by 120 LiDAR image if the end has not been reached.
19. If the end of the image has been reached, call the detectDitch() function to color all pixels that represent ditches or gaps in depth value between objects blue. The detectDitch() algorithm will be explained in more detail later in this document.
20. Output the final image as a JPG file.

Estimation of Ground Plane Using Linear Regression with Method of Least Squares

This section explains the algorithm used to determine the angle THETA that represents the angle by which the ground’s reference frame will be rotated to become parallel with the LiDAR’s reference frame. For now, let’s assume the LiDAR is upright (has a 0° angle of inclination). First, we need to estimate the ground plane using linear regression with the method of least squares to determine the slope of a completely flat ground plane as it appears in the LiDAR’s point-of-view.

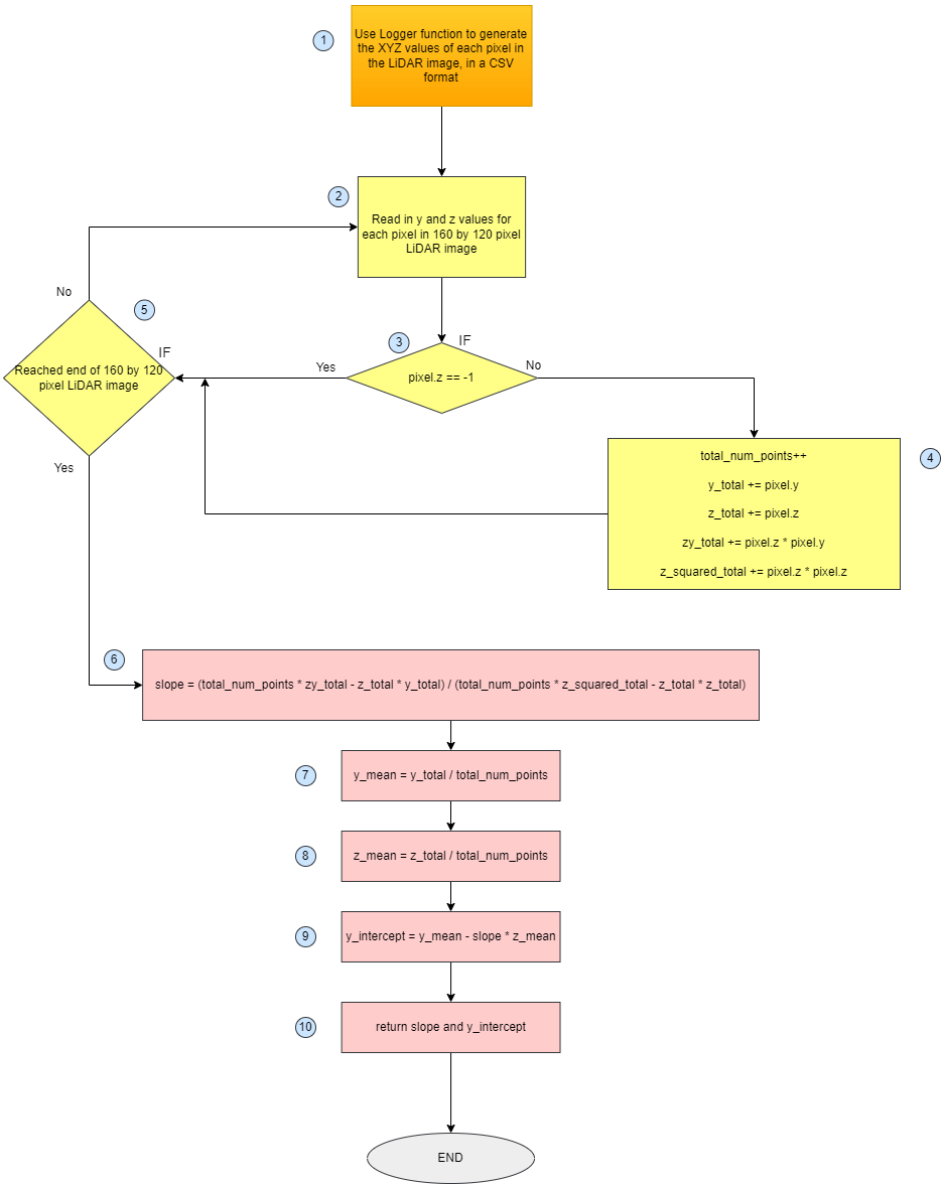
To determine the slope of a perfectly flat ground plane as it appears in the LiDAR’s point-of-view, I placed the LiDAR upright on the ground on a completely flat surface where only this surface and no other obstacles or

objects were present in the LiDAR's field-of-view. I then used the Logger function in the Sentis ToF API to generate the XYZ values of each pixel in this field-of-view.

Next, I created the linearRegression() function to read in these XYZ values from CSV files and perform linear regression on the y and z values of the pixels that have a valid depth value (the pixels with valid depth values are the pixels detected by the LiDAR that represent the ground plane), with the method of least squares.

The linearRegression() function outputs the slope and y-intercept of a line on this ground plane. The slope of this line represents the slope of a perfectly flat ground plane as it appears in the LiDAR's point-of-view. By taking the inverse tan of this slope and converting to degrees, we can determine the angle OMEGA by which the ground's reference frame must be rotated to become parallel with the LiDAR's reference frame when the LiDAR is upright. If not upright, we add the LiDAR's angle of inclination to OMEGA, to get THETA.

Here is a flowchart explaining the linearRegression() algorithm in more detail:



Here is a step-by-step explanation of each block of this algorithm:

1. Use the Logger function in the Sentis ToF API to generate the XYZ values of each pixel in the 160 by 120 pixel LiDAR image, in a CSV format.
2. Read in the y and z values for each pixel in the 160 by 120 pixel LiDAR image.
3. Check if the pixel has a valid depth value (z value of -1 means invalid depth value).
4. If the pixel has a valid depth value (z value != -1), add 1 to total_num_points (the total number of points on the ground plane that have been detected by the LiDAR), add the pixel's y value to y_total, add the pixel's z value to z_total, add the product of the pixel's z and y value to zy_total, square the pixel's z value before adding it to z_squared_total.
5. Check if the algorithm has reached the end of the 160 by 120 pixel LiDAR image and continue iterating through the rest of the pixels if the end has not been reached.
6. If the end of the LiDAR image has been reached, calculate the slope of a line on the ground plane.
7. Calculate the mean average of all the y values.
8. Calculate the mean average of all the z values.
9. Calculate the y-intercept of the line on the ground plane.
10. Return the slope and y-intercept of the line.

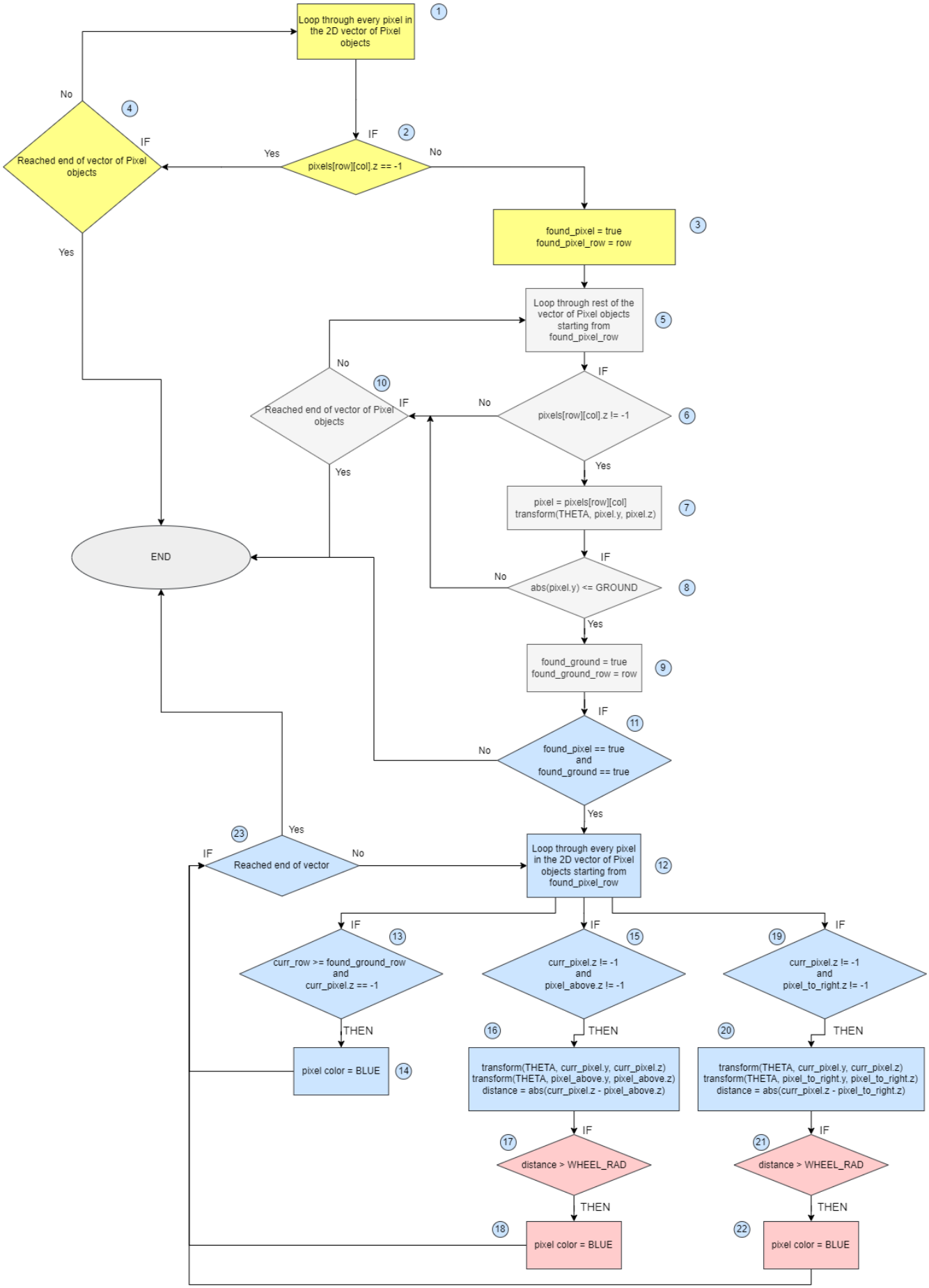
Algorithm to Detect Ditches and Depth Value Gaps

This section explains the detectDitch() function which colors all pixels that represent ditches or gaps in depth value between objects blue. The detectDitch() function takes in an image to perform the ditch detection on and a 2D vector of Pixel objects that represents all the pixels in the 160 by 120 pixel LiDAR image. The algorithm then loops through every pixel in the 2D vector of Pixel objects until a pixel with a valid depth value is found. Once a pixel with a valid depth value is found, the function stores the row of this pixel. Next, the function loops through the rest of the vector of Pixel objects searching for the first pixel on the ground that it finds and stores the row of this pixel.

Starting from the row of the first pixel with a valid depth value that was found, the algorithm loops through the pixels in the rest of the vector. Since the function had also saved the row of the first pixel on the ground that was found, every pixel in the vector that's below this row and has an invalid depth will be colored blue (every pixel on the ground that has an invalid depth can be considered a ditch).

Otherwise, the algorithm compares the depth values of the pixel the loop is currently on to the pixel directly above it and the pixel directly to its right to determine if there is a significant gap in depth value (a depth value gap greater than the radius of the Rover's wheels). If there is a significant gap, the algorithm colors the current pixel blue to indicate to the operator that there may be a crevasse at this point or that it could represent the edge of an obstacle that's in-line with another obstacle in the LiDAR's field-of-view.

Here is a flowchart representation of the algorithm:



Here is a step-by-step explanation of each block of this algorithm:

1. Loop through every pixel in the 2D vector of Pixel objects, searching for the first pixel with a valid depth value.
2. Check if the pixel has a valid depth value (z value of -1 means invalid depth value).
3. If the pixel has a valid depth value, set the found_pixel bool variable to true to indicate that a pixel with a valid depth has been found and set the found_pixel_row variable to the current row to remember the row of this pixel.
4. If the pixel did have an invalid depth value, check if the end of the vector of Pixel objects has been reached. If it has been reached, the algorithm ends. If it hasn't been reached, continue to the next iteration.
5. Once a pixel with a valid depth value has been found, loop through the rest of the vector of Pixel objects starting from found_pixel_row, the row of the first pixel with a valid depth value that was found.
6. Check if the pixel has a valid depth value (z value of -1 means invalid depth value).
7. If the pixel has a valid depth value, transform it from the ground's reference frame to the LiDAR's reference frame with the transform() function.
8. Check if the pixel is on the ground (has an absolute value height less than or equal to GROUND).
9. If the pixel is on the ground, set the found_ground bool variable to true to indicate that a pixel on the ground with a valid depth has been found and set the found_ground_row variable to the current row to remember the row of this pixel.
10. If the pixel did have an invalid depth value, check if the end of the vector of Pixel objects has been reached. If it has been reached, the algorithm ends. If it hasn't been reached, continue to the next iteration.
11. Check if the found_pixel and found_ground bool variables are both true. If they aren't both true, the algorithm ends.
12. If both found_pixel and found_ground are true start from found_pixel_row, the row of the first pixel with a valid depth value that was found, and loop through the rest of the vector of Pixel objects.
13. Check if the current row is below found_ground_row and if the current pixel has an invalid depth value.
14. If the current pixel has an invalid depth value and the row it's in is below found_ground_row, it means the pixel is on the ground has no detected depth value so it will be considered a ditch and will be colored blue.
15. Check if the current pixel and the pixel directly above it both have valid depth values.
16. If the current pixel and the pixel directly above it both have valid depth values, transform both of these pixels from the ground's reference frame to the LiDAR's reference frame. Then calculate the absolute value z distance between both pixels.
17. Check if the calculated distance is greater than the radius of the Rover's wheels.
18. If the calculated distance is greater than the radius of the Rover's wheels, there is a significant depth gap between both pixels and therefore the current pixel will be colored blue to indicate this gap.
19. Check if the current pixel and the pixel directly to its right both have valid depth values.
20. If the current pixel and the pixel directly to its right both have valid depth values, transform both pixels from the ground's reference frame to the LiDAR's reference frame. Then calculate the absolute value z distance between both pixels.

21. Check if the calculated distance is greater than the radius of the Rover’s wheels.
22. If the calculated distance is greater than the radius of the Rover’s wheels, there is a significant depth gap between both pixels and therefore the current pixel will be colored blue to indicate this gap.
23. Check if the end of the vector of Pixel objects has been reached. If it has been reached, the algorithm ends. If it hasn't been reached, continue to the next iteration.