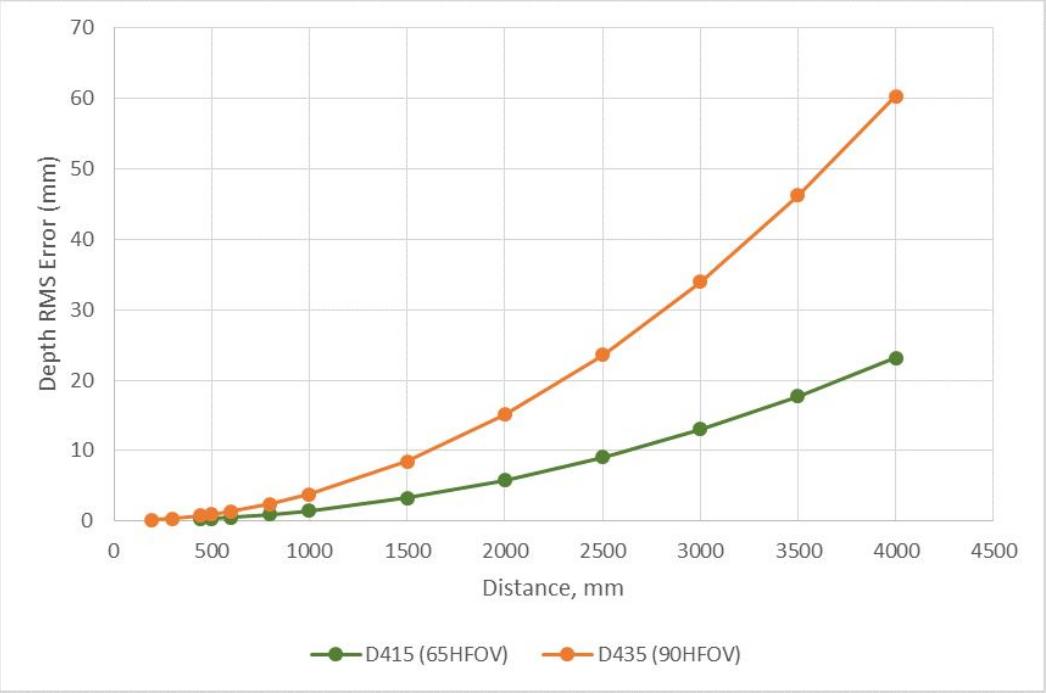
Traversable Surface Perception Report

a. Data and Preprocessing

The data consists of a *rosbag* collected from a D435i camera mounted over a robot vehicle being driven around campus (JHU) - hence the traversable surface primarily consists of (red) brickway (alongwith road, marble and if needed, grass lawns). The data consists of color and depth images which are by design of the camera unable to be perfectly aligned since they are not taken at the same time, but given the velocity that the vehicle can achieve, this synchronization error is negligible.
However, the depth estimation is very noisy and given the small distance of error constrained depth estimation, a robust calibration of the cameras and preprocessing of the images is *necessary* for the later part of the algorithm.



Please refer to this page for the calibration process.
The general depth filter outline is:
Depth_to_disparity -> Spatial_Filter -> Temporal Filter -> Disparity_to_Depth -> Hole Filling

Please refer to this white paper for further information on the topic.
Additionally, one may choose to remove parts of image which are further than a specified distance in the image (e.g. > 10 meters) as the depth estimation (for D435i) becomes extremely error prone over longer distances. This also reduces the number of pixels that we need to search to find the traversable path and helps us in filtering out major components of the image which are not useful to us (e.g. the sky). One can also filter out parts of image dependent of how noisy the depth estimation is at shorter distances which can happen due to irregularly formed object like dense tree branches or rugged hole in the path (it may happen due to a bad detection due to

glare and lighting issues) - it may be a good practice to ditch a frame if the depth detection is extremely poor even after filtering.

b. Algorithm Implemented

Figure 1 provides a basic understanding of the traversable path detection (TPD) algorithm.

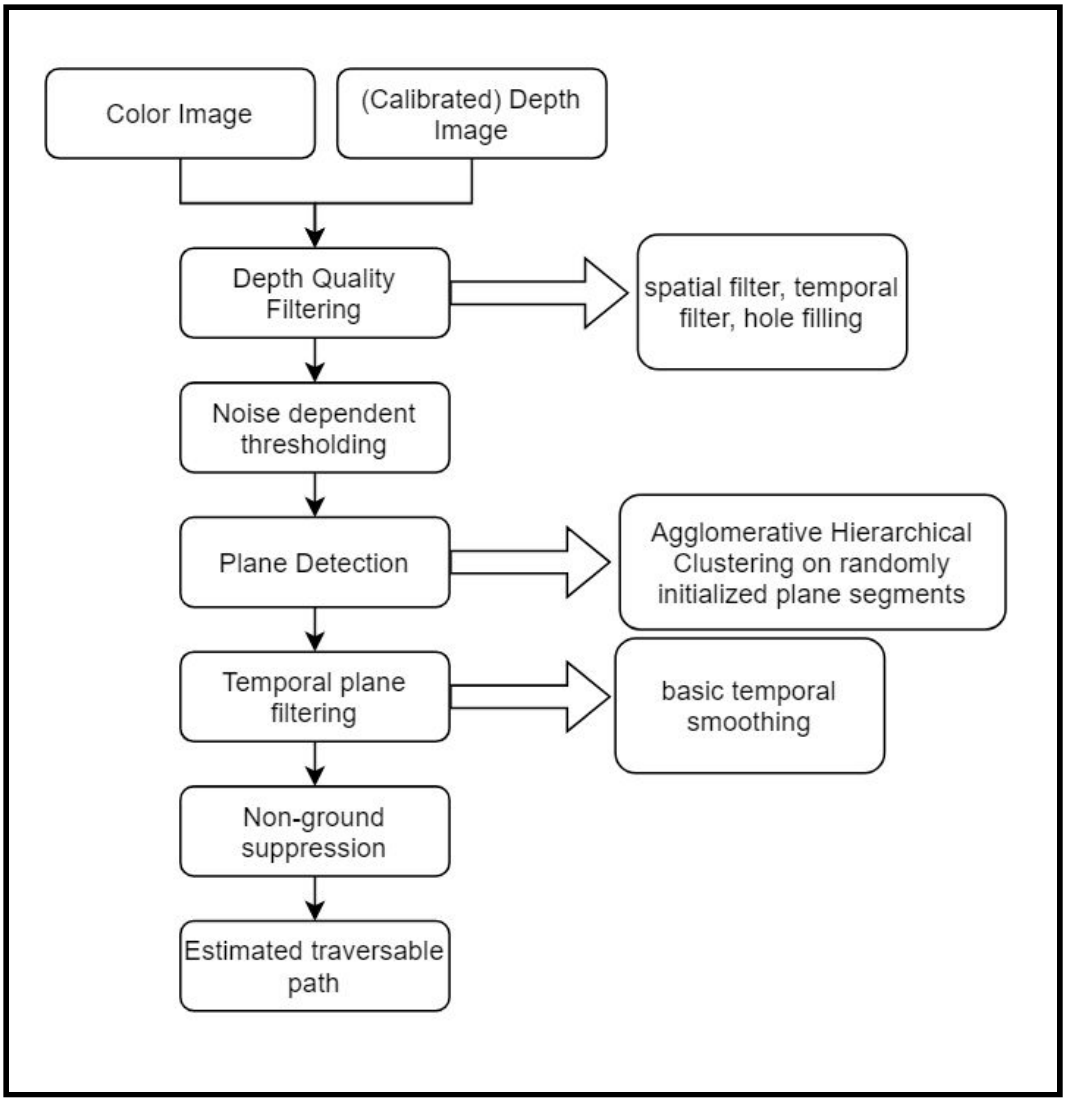


Figure 1. TPD Algorithm

c. Plane Detection

The plane detection has two major parts:

1. Over segmented texture detection
2. Ground Plane Detection

In order to further reduce the area of the image where plan to search the traversable path in the image as well as to get a more confident detection, we

run an approximate/unsupervised segmentation algorithm on the color image and get an output which has over-segmented the traversable path. This is a pretty strong assumption that we are making, but has been proven to be quite robust in practice, since the traversable in question is usually available in a repeatable textured manner - brickway, grass, concrete, marble, asphalt, dirt. In our implementation, we use a HR-Net with PPM decoder which has been pre-trained on ADE20K dataset - which seemingly provides us with noisy image segmentation which encompasses the actual traversable path.

For plane detection, we primarily use to approaches - RANSAC based and Agglomerative Hierarchical Clustering - using the depth and color image to calculate the surface normals in the image and cluster the point cloud in different space - euclidean, surface normal, color, geometry, etc. Most of these approaches include a random sampling of initial points to calculate the surface plane parameters. Though, we do not have an implementation which involves plane center tracking - in order to have a much more robust detection, we can choose to track the center of the detected plane from the previous frame and constrain our random initial point selection around a small radius of the previously detected plane center coordinate. This would require a provably robust first frame plane detection which can be done by
i.) ensuring the first frame is largely the traversable path
ii) taking an ensemble of multiple run of the plane detection algorithm

Since the depth estimation is pretty noisy, it constrains the plane detection robustness, and we will notice small blobs in the detected plane which can actually be removed via per pixel temporal tracking (which has not been implemented yet) or taking an ensemble per pixel output of the plane detection algorithm (i.e. if a pixel was a part of a plane which is centered at (x,y) and the same pixel is part of a similar plane centered at (x+e, y+e) where is e is small in most of the runs of the algorithm, then the pixel is part of the ground plane).

It is optional to use the probabilistic output from the segmentation model as well as the percentage from the ensemble output to generate a confidence map instead of a binary map for the traversable plane. A simple example would be just taking the mean of the probabilities - for e.g. if a pixel has a .90 probability of being a road in the segmentation model output and if it is being detected as a plane pixel in the plane detection model 3 out of 5 times, then we can consider the traversable path confidence for the pixel to be (.9 + ⅗)/2 which is 75%. Further confidence calculation rules can be added depending on the class of the segmentation model as well the confidence of neighborhood pixels.

It is a good practise to track the delta between the sequential segmentation and plane detection output. A significant change in detected area reveals a bad run of the respective algorithm - which we can either re-run if it has a stochastic component (Plane Detection) or dump the output in case its determinate (deep network segmentation).

d. Postprocessing and Temporal Tracking

e. To-Do

i. Plane Center tracking.
ii. Constraining random point selection in plane detection around a small radius centered around previous frames detected ground plane center.
iii. Parallelizing of the algorithm for scaling up the fps.
iv. Per pixel temporal tracking
v. Confidence Map calculation instead of binary mask output