Rover Path Navigation via Aerial Imagery

Dyllon Dunton
University of Maine
dyllon.dunton@maine.edu

Sophie Walden
University of Maine
sophie.walden@maine.edu

Abstract

Automatic navigation is a field that is hard for land-based rovers because they are only able to scan their immediate surroundings. This paper proposes an alternative solution using drone footage of the location to create a 3D scan of the environment. This scan includes using a depth estimator ZoeDepth to create a heightmap and a self-built UNet model to create a segmentation map. The segmentation map describes different surfaces the rover might interact with such as vegetation, water, roads, and more. From this, we created a pipeline where you can supply a single mp4 of top-down drone footage and we are able to create a 3D interactive visualization for you to see a rover's path through the footage.

1. Introduction

An annual University Rover Challenge from the Mars Society brings robotics teams together to make new systems to guide rovers on Mars terrain. In recent year's competitions, a couple of teams made a new augmentation to their strategies: using an aerial drone to accompany the rover with the sole purpose of aiding path planning. This allowed the teams piloting the drones to plan an optimal path for the rover around challenging terrain. What this paper proposes comes from a similar approach, but aims to work with top-down drone footage to make explorable visualizations of the mapped terrain to guide these teams with making planning decisions. This is a precursor to a system that will automatically guide rovers along the optimal planned path.

2. Mapping and Machine Learning

2.1 Video Stitching

The primary function of this system is to create three maps in addition to a planned path. To turn the input drone footage into a map, the team has used OpenCV to stitch the frames together into a rectangular panoramic photo. This is done by using a function that finds a homography between successive frames of the video and stitches them into a large panoramic photo. This RGB map is the first of 3 maps generated.

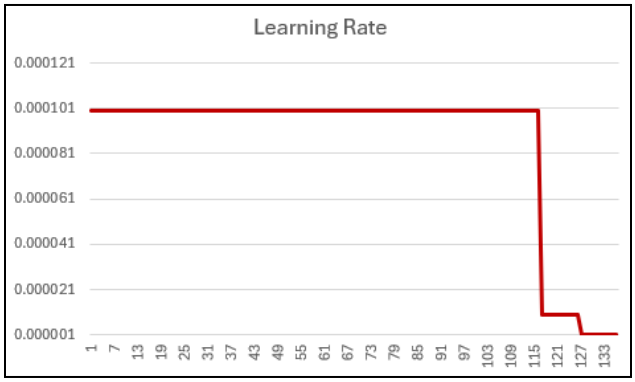
2.2 Depth Mapping

The second map generated is an estimated topological map, where each pixel is an estimated distance from the camera of the drone. Using the ZoeDepth model from [3], the team is able to generate this map, and normalize the pixel values to be on an 8-bit scale.

2.3 Segmentation Model

The third and final map is a semantic labeling of the terrain, which can determine which parts of the terrain are ideal, unideal, and undrivable terrain. The dataset [1] consisted of 400 densely labeled, 4K resolution images, with 23 different terrain classes including pavement, grass, dirt, water, trees, etc.. The team was able to generate 12,400 images with their corresponding labels, from patches of the large images of various sizes, all scaled down to 224x224 resolution. This resolution was chosen since it is the input to the segmentation model we designed. This model is built on the Resnet34 model, which is a 34 layer convolutional neural network that acts as an image classifier. Using the convolutional encoder from this model, the team was able to append a convolutional decoder to scale the image back to the output size of 224x224. This was done to create a U-NET [2] architecture which consists of a pair convolutional encoder and decoder. The dataset was split into a ratio of 9:1, where 90% of the images were to be used for training, and the remaining 10% for validation. The model was trained using the Adam optimizer with a learning rate of 0.001 and a categorical loss function with a a batch size of 32. After each epoch of training over the 11,160 training images, the model was run on the validation set of images, and an accuracy metric was calculated by determining the percentage of pixels classified correctly. After 136 Epochs of training, the loss had converged and the validation accuracy was measured to be 91.683%. The inaccuracies of this model were all along borders from one semantic label to another, meaning that it was

determining the general areas of the classes, but had sloppy borders. However, this is okay for the current project since we only need to know the general areas to avoid. Previous attempts to train the model involved simply resizing the 400 images to the desired resolution, which only reached a validation accuracy of approximately 76%. This further shows that the new method works well to improve the accuracy of the model. The figures below show the learning rate, training loss, and validation accuracy across the 136 epochs respectively.



S

The team supervised the validation accuracy and training loss, and manually adjusted the learning rate to prevent overfitting, starting with 1E-4 then 1E-5, and finally 1E-6. The training was stopped at epoch 136 since the validation accuracy was no longer trending upwards and the training loss continuing to lower would have caused overfitting

2.4 Semantic aggregation

To use this model to create a semantic map, the variably sized rgb map is split into many 224x224 patches. Running each patch through the model and aggregating them back into a map would create a map with inconsistencies around patch borders. To combat this, the patches were overlapped by 80% which created 5 maps with borders at every 44 pixels. Taking the mode of each pixel in the maps allowed for a voting system that remedied the inconsistencies around the original patch borders, and created smooth borders from one semantic label to the next. From this we now have 3 maps to be used in visualization and path planning.

3. Visualization

The upside of the machine learning models is that just from a short video they can generate a massive amount of data. For example, a 10-second clip of a drone flying over a residential area produced a 3000x1000 height map and biome segmentation image. If we were to tackle areas that take up to a minute of scanning we are looking at 18000x1000 maps. That is why a major portion of this project committed to was just the visualization tools. Ultimately, we created an interactive demo website that can instantly take the model outputs and show them on a deployed site for users to interact with.

3.1 Pathfinding

One non-machine learning part of the project that needed to be created for the visualization was pathfinding. For this a custom implementation of A* was created that took in the heightmap and segmentation map and spit out an optimized traversal. The only difference between normal cost functions is that we used values for heightmap and segmentation.

The first value calculated in the cost function, g(x), is the distance away from the origin. This is calculated by using the Euclidian distance formula extended to use the heightmap pixel value as the third coordinate.

$$g(x) = \sqrt{(x_1 - x_o)^2 + (y_1 - y_o)^2 + (h_1 - h_o)^2}$$

Then a similar process is applied between the point and the target cordinate. This is a good heuristic for the approximate distance between any given point and the end target.

$$f(x) = \sqrt{(x_1 + x_t)^2 + (y_1 + y_t)^2 + (h_1 + h_t)^2}$$

Finally, these functions are combined into our cost function. In this function, we also take the segmentation class value, which is constrained down to the greyscale values 0-240, to add a multiplier to encourage driving on optimal biomes such as roads as opposed to houses. The reason it is 0-240 is because the classes are condensed down into 3 different classes: optimal, semi-optimal, and nonoptimal. The segmentation

map is turned into a greyscale map with value of 0 to represent nonoptimal, 120 for semioptimal, and 240 for optimal.

$$C(x) = f(x) + g(x) * (241 - segmentatoinClass)$$

These functions, while not exactly constraining to the usual A*, were an approximate path planning algorithm to create optimized paths with the additional data. The output of the pathfinding was a JSON file with exact points from the planned path which is then plugged directly into the visualization.

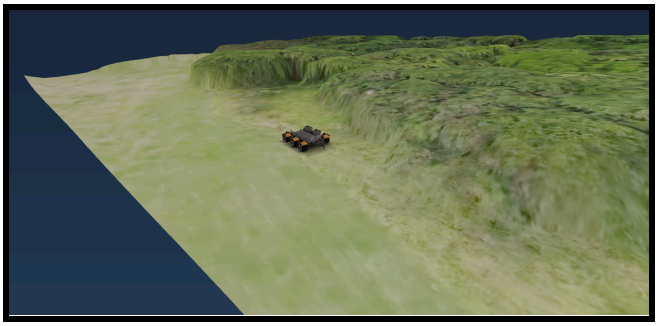
3.2 Visualization Website

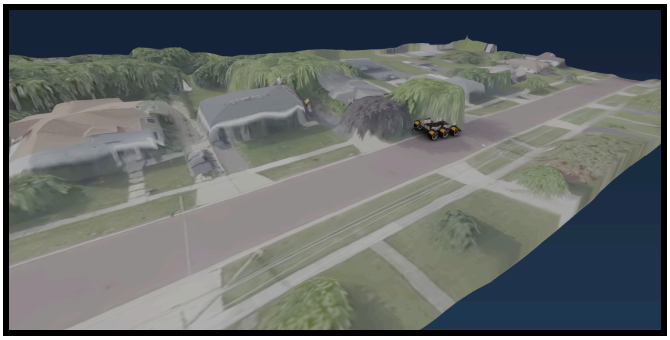
For visualization purposes, we used a 3D javascript library ThreeJS to create interactive displays. ThreeJS is a JavaScript library that allows you to display 3D models directly in the browser. For our specific visualization, it takes the computed heightmap from section 2 and creates a displacement map as the base geometry. Then on top of that it overlays the visuals of the original image. From this alone, it creates a 3D display of the visuals from the video.

The next step is to give the users interaction within this generated model. To do this a camera with flying controls was added to the canvas holding the model. This allows the user to move and see the generated data displays from multiple different viewpoints. The result of this looks very similar to Google Street View except we took the cost from a 360 camera to just having a top-down mp4 of the land.

The pathfinding is integrated into each visualization directly supplied from the JSON files described in section 3.1. To display it a model of a rover is rendered into the environment which drives along the path generated. Since the pathfindings output is the rover's path as 3D points, the rover can be shown traversing vertically over obstacles. Additionally, a setting is added that allows the user to attach the camera to the rover. This lets us show off the view of the rover as it traverses through these generated models.

To integrate this as much as possible into the models all it takes to add a new display is to upload new height maps or images. This is stored in an array in the component for model exploration. From here the user is given the option to swap live between the different visualizations. This also swaps the loaded path allowing you to see the rover pathfinding through multiple environments. All of this, combined with a homepage explaining our methods, can be found at: https://bit.ly/cos473



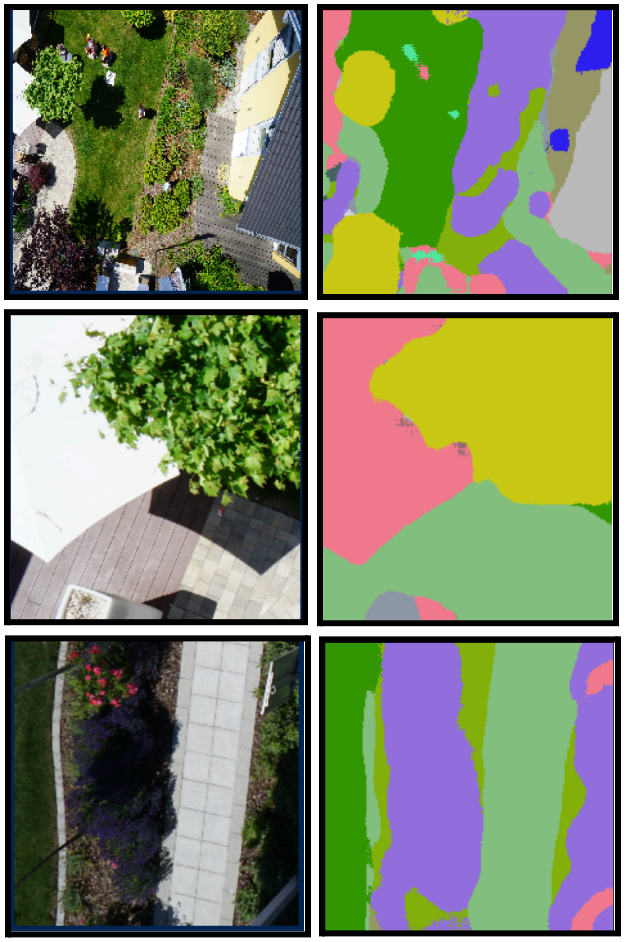


4. Complete Pipeline

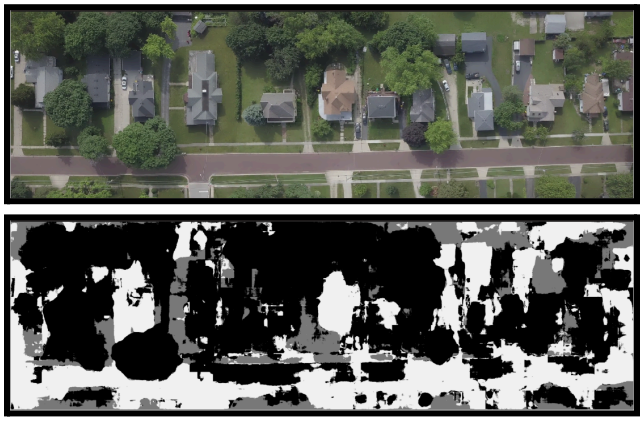
The functioning of this system is contingent on the integration of all of the different subsystems. Fusing image stitching, depth mapping, semantic mapping, visualization, and finally path planning is essential. To make this work seamlessly, the team modularized each program in their separate files into functions that could be called from the main program. This will then call each program to stitch the video into a panorama, create a depth map, run the segmentation model on patches of the image, aggregate the patches together, use that information to create a path using A*, and then use the outputs for the web-based visualizer. The source code for the project can be found at: https://github.com/DyllonDunton1/Height_Map_UNET

5. Results

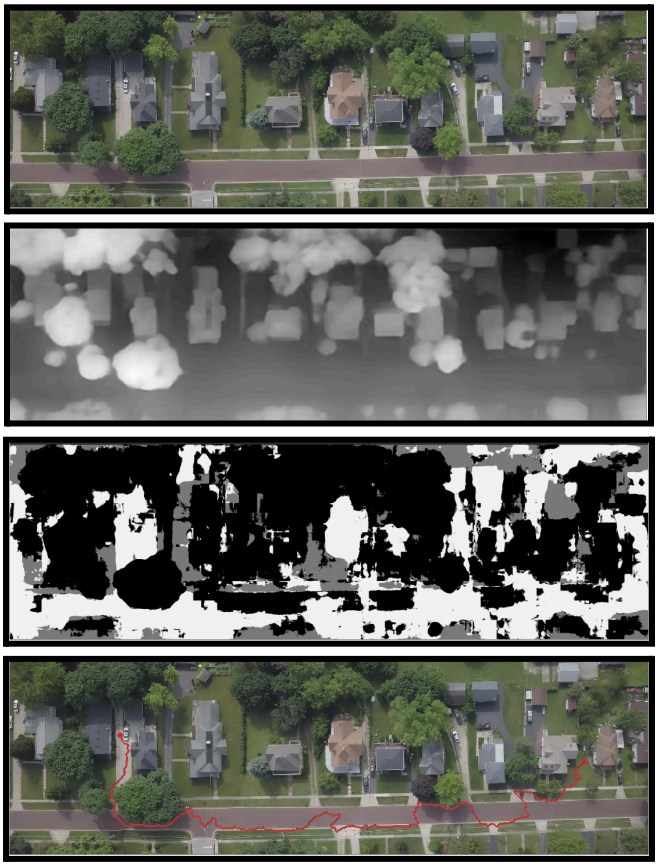
This system has shown excellent results. As said in Section 2.3, the segmentation had a validation accuracy of 91.683%. Examples of validation test images are in the figure below, aligned with the test image on the left and the predicted semantic labels on the right.



It is clearly seen from the test that the model is generalizing moderately well, but loses accuracy around the borders of the classes. Results from aggregation of the segmentation model and determining drivability can be seen in the figure below. The first image is the stitched panorama, and the second is the aggregated drivability map where white is ideal, gray is drivable, and black is undrivable.



The voting system worked well to reduce boundary inconsistencies at each patch, and made much smoother semantic predictions. Putting everything together, the model is able to make a decent path through the different landscapes. This is shown in the figure below. The first image is the stitched panorama, the second is the depth map from the ZoeDepth [3] model, the third is the drivability map from semantic aggregation, and the fourth and final image is the path determined from A* overlaid onto the stitched image.



This test worked very well. The stitching and depth map were created exceptionally well, but it can be noted the the drivability map is missing some key information. Mainly it can be seen that the road was being classified as an obstacle, and some of the houses were determined to be pavement, which is an ideal driving surface. Though, since the depth map worked well, it can avoid the stark changes in height from the houses when path planning. Further discussion of the results will be conducted in the next section. More tests and their results can be found in the Appendix.

6. Discussion and Future Work

There are many short points in this system. In this section, the team will address those points and suggest improvements for future work. To improve this project, several aspects need to be reworked. The following improvements would be implemented: Hybridize the segmentation system, implement depth and segmentation information more effectively, and fine-tune the cost function for path planning. The depth estimator model works well for this application, so it should not be changed.

6.1 Hybridized Segmentation

When assessing the inconsistencies with the segmentation model, the validation accuracy was compared against similar models built on the same dataset. This was done by looking at the "code" section of the Kaggle dataset and examining the works of others. It became clear that our 91.683% validation accuracy was on the very highest end of the spectrum of models. This means that though the methods used to create the model were sound and insightful, the actual shortcomings were coming from the dataset itself. The "Semantic Drone Dataset" [2] did not have enough variety in features to generalize well enough to images outside of the dataset. Even though the accuracy was measured on a completely different set of images than which were used for training, it seems they were all in the same area, which creates problems such as new roads not being classified as pavement since they are not pale and cracked, and grass not being classified when it is patchy and unsaturated.

To solve this issue, It became evident the team would need to find a new dataset. In fact, research shows that it is more effective to train many models with less classes each. This allows for better generalization for each class. However, problems will arise if we simply make binary classifiers for each class. Each one will likely have higher accuracy, but aggregating the models together will cause conflict in similarly colored and textured classes. For example, the "pavement" and "house" classifiers could both say a house with asphalt shingles is pavement. In this case, choosing preference towards either would eventually cause issues with allowing the rover to drive into obstacles or obstruct the optimal path along the pavement. To fix this, I propose a hybridized approach where similar classes are put into classifiers together. This would create 3 models. The first model will classify trees, vegetation, grass and other forestry. The dataset from [4] would work well for this application. The second would classify buildings, walls, roads, bridges and other manmade structures. The final would classify water, dirt, sand, and other soil types. These three models would allow for better generalization amongst the various classes as well as reduced conflict amongst the said classes. Preference would be given in an order befitting of the location. If it is an urban area, the man-made classifier would be preferred. However, if the area is a forest, then the vegetation classifier would be preferred. A Large Language Model (LLM) like GPT-4o could be used to determine the location type so that the program can determine the preference order. It is also possible to make a categorical classifier for the biome type which would take more time but cost less in the long run and be more suited to this project.

6.2 Depth and Segmentation Implementation

To improve the implementation of these two models, it is important to assess where they are falling short now. The main issue with the current implementation is that it creates a jagged, nonoptimal path that doubles back on itself at times. This is not due to the choice in path planning algorithm, but because we use change in depth as well as the drivability (derived from the segmentation) as a cost for each step. Since the depth model has a lot of gradients, the model makes drastic turns to avoid a steep path where it is not totally needed. Though it is promising to have the shallowest gradient as possible, a short and smooth path is also important.

I propose two improvements to this system. The first is to create an absolute depth map. This can either be done using a rover of unknown size in the image to scale the relative depth, or using a Time of Flight (TOF) camera or LIDAR to determine an absolute depth map. The second improvement would be to use the depth readings differently. Rather than use it as a cost for each step, taking the gradient of the depth image at each pixel will give us the slope at each point. Using a maximum gradient variable would allow the team to create a mask of undrivable areas that are too steep. With this map, we can find the areas flat enough to drive on. Then we can use the path planning algorithm normally, which will find the shortest path in this area. We can use the segmentation map in a similar way, making the drivable area the intersection between the flat and drivable zones. If no path is possible, then the restrictions will relax until it is possible to find a path.

6.3 Other Path Planning Algorithms

While A* is a widely used planning algorithm, the team believes exploring additional continuous path planning algorithms like RRT* would be more advantageous to finding the way through thin flat alleys more quickly. I propose using this in the future. However, this should be the last improvement since A* works sufficiently well for the time being.

6.4 System Utilization

Currently, this system is only to be used as a tool for making the most optimal path but does not inform the rover of how to use the path automatically. Two possible methods include using photogrammetry techniques to create a synthetic video from the perspective of a rover along the path and using a machine learning model to take actions that make its viewpoint look as similar as possible to the video. The second would be to have the drone hover far above and relay commands to the rover to make sure it is following the planned path. For now, neither of these has been implemented, but it is clear that the first is advantageous for long trips since it conserves energy by allowing the drone to land, while the second is advantageous for short trips since it will be much quicker.

7. Conclusion

The project resulted in an excellent proof of concept for a UAV assisted path planning system. There is much work to be done to improve this system, but it shows great promise and should show optimal results when improvements have been made.

References

[1] **A. W. Saf**, *"Semantic Drone Dataset"*, [Online]. Available: https://www.kaggle.com/datasets/awsaf49/semantic-drone-dataset. [Accessed: 24-Nov-2024].

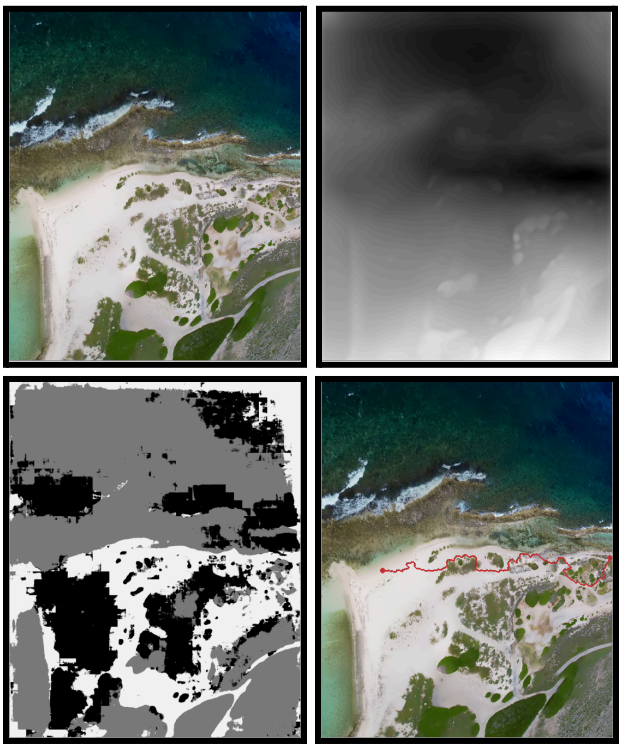
[2] **O. Ronneberger, P. Fischer, and T. Brox**, *"U-Net: Convolutional Networks for Biomedical Image Segmentation,"* presented at the *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, Munich, Germany, 2015, pp. 234–241. [Online]. Available:https://arxiv.org/abs/1505.04597.

[3] **S. F. Bhat, R. Birkl, D. Wofk, P. Wonka, and M. Müller,** "ZoeDepth: Zero-shot Transfer by Combining Relative and Metric Depth," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1-10, 2023.

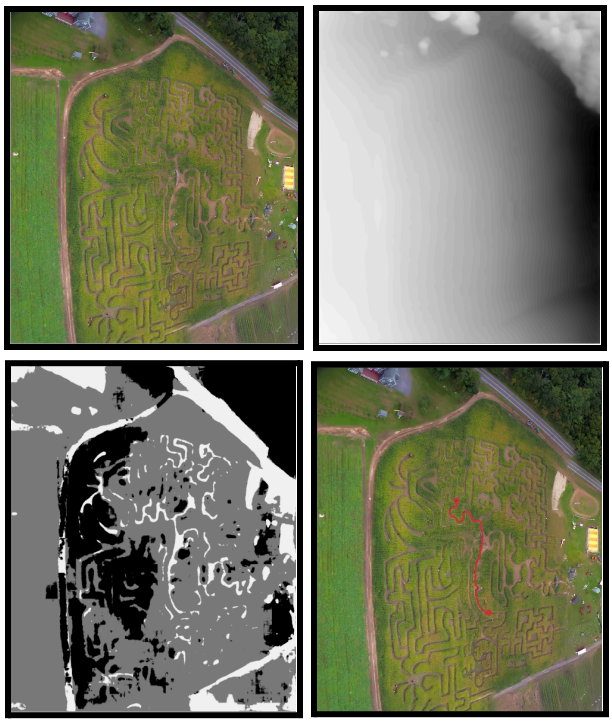
[4] H. R. Flø Rasmussen, J. H. Rosenlund, and K. D. Madsen, *"Forest Inspection Dataset for Aerial Semantic Segmentation and Depth Estimation,"* arXiv preprint arXiv:2403.06621v1, Mar. 2024. [Online]. Available: https://arxiv.org/abs/2403.06621v1. [Accessed: 24-Nov-2024].

8. Appendix

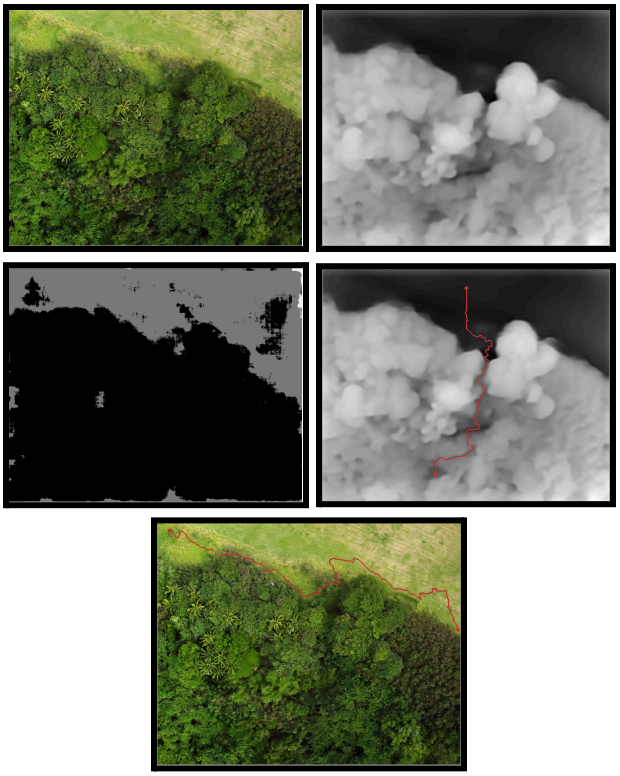
The pipeline was tested on many different videos to test the generalization of the system. The first was a beach. The results are in the figure below.



The model had a hard time determining that the water was undrivable. However, the rest of the surfaces were labeled with high accuracy and the path was generated well. The next test was at a corn maze. We knew that this would be a good test for the segmentation model. Since the maze walls are very short, the depth map will not end up helping very much. This means that the brunt of the work would fall on the segmentation model determining where the rover can drive in the maze.



The drivability was able to be determined rather well when the path was relatively thick. Because of this, it was able to find approximately 60% of the path. As expected, the depth model did not prove useful. Nevertheless, the path generation performed exceptionally on a portion of the maze that was found. The next test was in a high vegetation area. This included a field on the edge of a forest. This was mostly to test to see if the drivability would be able to distinguish between the drivable grass and the undrivable trees and bushes. The results are in the figure below.



The drivability model worked well, noting that the grass was drivable but not ideal, and the trees and bushes were undrivable. The first path was generated to test the depth map to see if it could find the path of only bushes that would need to be trekked to enter the forest. The second path took the rover from one side of the field to the next without entering the forest. With a total of four tests conducted, the model was determined to be a satisfactory proof of concept and is generally useful at its current state.