CIS 5603 Project Report

Rover Maneuvering AI Logic

Anku Aravind and Arnab Dey
Temple University
May 2021

Abstract—The intent of this project work is to emulate the behavior of autonomous planetary rovers by using machine learning algorithms for evaluating terraneous surfaces both from satellite images and from on-board camera images, and then applying AI based search algorithms to facilitate movement of rovers over the surface based on different constraints – just in the similar way a robot navigates. It is simply an academic evaluation of these algorithms that may be applied for different sub-tasks in the navigational process, rather than a deep-dive into the technical complexities dealt with by space agencies, and is based out of publicly available images and research papers.

I. Introduction

In a number of instances different Mars rovers like Spirit, Opportunity and Curiosity have been observed to get stuck in the sandy dunes while maneuvering over the terraneous surfaces, which resulted in significant loss of energy and navigation time [20]. Specifically, Spirit had to end its mission thus. Each rover has limited lifespan in terms of Sols or Martian days and there is a limited communication bandwidth (80MB / Sol) between the control center on earth and the rover. Since these rovers are solar powered the time of the Sol at which the navigation happens and the extent of exposure to sunlight also matters. Hence it is crucial for the rovers to use ML / AI algorithms to navigate autonomously for capturing more information at the optimal usage of energy. Worthwhile mentioning here SPOC-Lite [21] [22] and VeeGer [23] are publicly available projects from NASA's Jet Propulsion Laboratory for terrain classification and terrain-based navigation (driven by solar power) respectively. Athena [22] is a test rover from JPL using similar maneuvering techniques.

We have attempted to use Deeplab algorithms for the image classification tasks – both from sample satellite images for global planning of the navigation process as well as from on-board camera images for local planning during the movement. Search algorithms like A* and D*-lite are evaluated later in order to find the path of the rover over different terrains.

II. Semantic Segmentation of Martian Terrain

The rover utilizes two planners - a global planner and a local planner to navigate from the source to it's destination. Global planner is designed to obtain the initial heuristics and produce a coarse path from a segmented(terrain classified) satellite(in our case HiRISE) image. The local planner segments the image produced by rover's on-board camera to avoid unfavorable terrain for navigation, which fine tunes the path. Semantic segmentation is at the heart of terrain classification.

A. Data for Global Planner

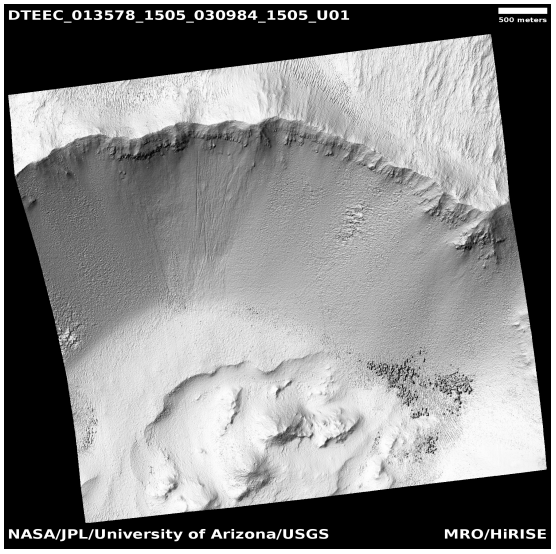
Global planner require satellite or top view images of the terrain. Figures 1 and 2 are sample images obtained by HiRISE [1] on Mars Reconnaissance Orbiter [3]. NASA uses a deep learning model SPOC(Soil Property and Object Classification) [4] to classify the terrain on these images. NASA has used thousands of HiRISE images to train SPOC. The challenge to build and train a deep learning model from HiRISE images is as follows.

- *Pre-processing:* Actual HiRISE images are very huge(hundreds of MB, depending on the location) and requires a lot of pre-processing

to even begin labelling. Cropped images are available, but that pose other challenges.

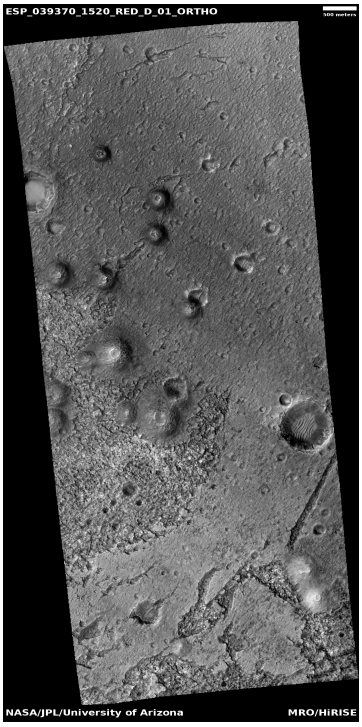
- *Understanding HiRISE images:* Consider Figures 1 and 2. We found it very difficult to understand and identify the features in these image. Besides identifying Figure1 as a crater, it's difficult to identify features such as sand dunes, plain, rocks and other terrain features in these images.
- *Labelling the image:* Since understanding the image itself is difficult, it is impossible to label HiRISE images. Besides, from my experience I find it confusing and erroneous.
- *Lack of labelled data:* NASA hasn't released any labelled Martian terrain data which would be useful for training ML(machine learning) or DL(deep learning) models.
- *Model constraints:* We used $DeepLabv3+$ [6] for semantic segmentation and it requires color images for training [5]. HiRISE images are gray scale and the available color images are artistic renderings.

Because of these challenges, we proceeded with the assumption that terrain classified HiRISE images are readily available instead of developing a DL model to train and classify them.



Source: NASA/JPL/University of Arizona

Fig. 1: Sample HiRISE Image



Source: NASA/JPL/University of Arizona

Fig. 2: Sample HiRISE Image

B. Data for Local Planner

Local planner uses images obtained by the rover. Figure 3 is a sample image taken by Mastcam [8] of Mars Curiosity Rover [9]. 110 Mastcam images [7] [10] from Mars Curiosity Rover were used to train a DL model for semantic segmentation. Lack of labelled data was the only reason to limit just 110 images for training and validation - all 110 images were labelled manually in Adobe Photoshop [11] for training. The $training : validation$ split is $101 : 9$. All the images were obtained from Curiosity MSL Analyst's Notebook [7] - sols 2926 through 2933. The reason to use Photoshop for labelling are:

- *Noise reduction:* We're labelling landforms that has contours which would be difficult with regular labelling tools. This was under the assumption that reduced noise would bring better performance.
- *Lack of domain knowledge:* We're new to semantic segmentation and wasn't aware of labelling tools. Our prior experience with Photoshop was the motivation to use it.
- *Open source tools:* While open source tools are great for community, it doesn't provide any

guarantee. Besides, installation and resolving dependencies could be difficult.

1) Labels: We divided the Martian terrain into 5 sub-terrains based on the navigational convenience of rover.

- *Terrain 1:* Best traction, mostly plain and has a few to no obstacles.
- *Terrain 2:* Traction is good and has some small stones.
- *Terrain 3:* Traction is sufficient, mostly ridges and rocks.
- *Terrain 4:* Traction is low to none, fine sand.
- *Terrain 5:* No traction, sand dunes.

Terrain 1 is the best and Terrain 3 is the least favorable for navigation. Terrain 4 and 5 are unfavorable. Figure 5 shows the labels color map of labels.

Note: The terrain was classified based on our understanding of the images used for training. This classification may not match scientific classification.

Source: MSL - Curiosity

Fig. 3: Original image

Fig. 4: Label for Figure 3

Fig. 5: Label color map

C. DL Model

We used $Xception(Xception65)$ backbone in $DeepLabv3+$ [6] to train and produce semantic segmentation of Mastcam images for local planner. We had 6 classes to segment - 5 classes(terrain 1 through 5) and background(sky).

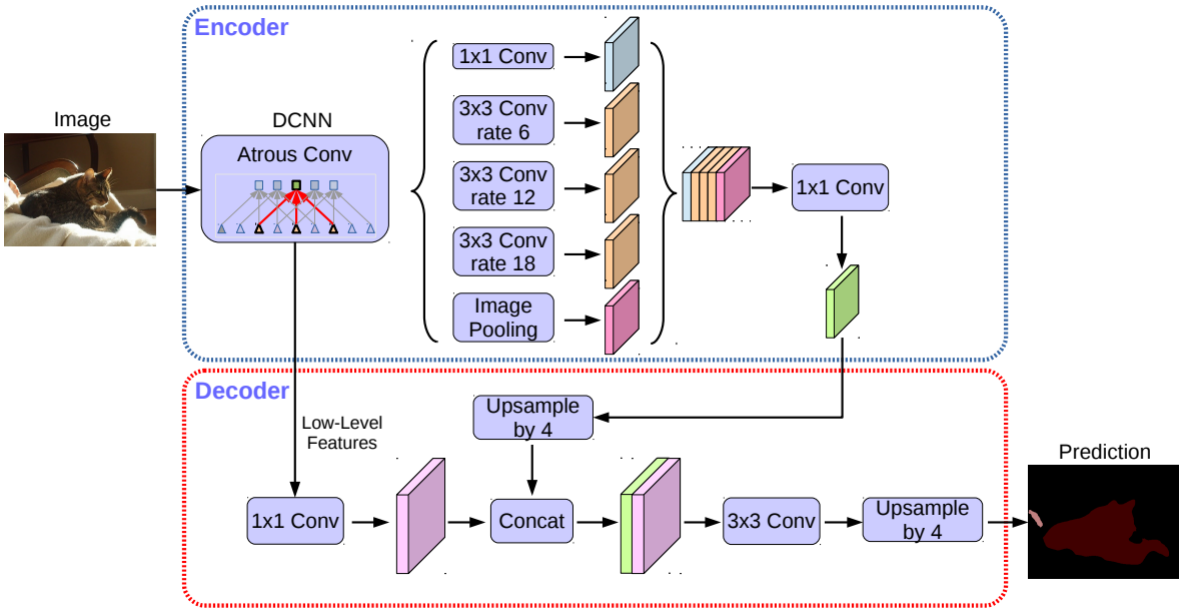
DeepLabv3+: DeepLab is a computer vision project with the goal of segmenting every pixel in an image [6]. $DeepLabv3+$ is the fourth version of this project. It uses Atrous Spatial Pyramid Pooling(ASPP) along with an encoder and a decoder to segment images, as shown in Figure 6. The reasons to use $DeepLabv3+$ are:

- NASA's SPOC is based on DeepLab [18].
- Accuracy is great - 89% on PASCAL VOC 2012 dataset [14] and 82.1% on Citiscapes dataset [24].
- Approach 1 failed.

We tried three approaches to produce semantic segmentation.

1) Approach 1 - Build a model: The original plan was to build and develop a new model for semantic segmentation. This was challenging because of the lack of expertise in developing ML and DL models for image processing. Besides, training data was very less. So, this approach was discarded.

2) Approach 2 - Use $DeepLabv3+$ to train a new model: $DeepLabv3+$ can be used to train a network from scratch. We tried this approach and failed. We attribute this failure to our configuration of $DeepLabv3+$. Unfortunately, $DeepLabv3+$ doesn't have proper documentation and search for support and clarifications proved futile.



Source: DeepLabV3+ [6]

Fig. 6: DeepLabV3+ architecture

Figure 7 shows the initial labels of image from Mastcam shown in Figure 3. This color map was designed to be used in this approach. These initial labels were converted to the label color map in Figure 5 to be used in *Approach 3*.

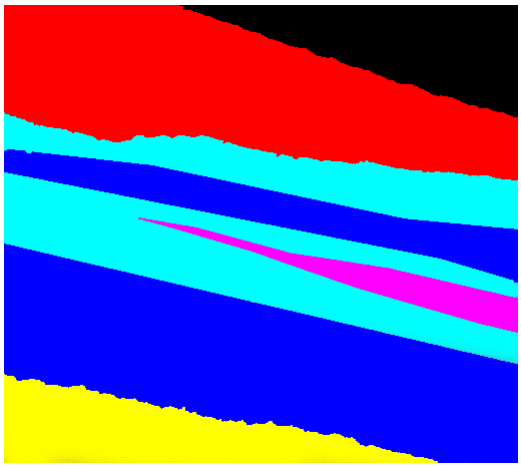


Fig. 7: Planned label for Figure 3 to be used with *Approach 2*

3) Approach 3 - Transfer Learning: This is the approach implemented in this project. We used the checkpoint of pretrained $Xception65$ backbone($xception65\_coco\_voc\_trainaug$ [15] - trained on PASCAL VOC [14], COCO [13] and ImageNet [12] datasets) to initialize our training of Mastcam images.

We were skeptical of this approach because PASCAL VOC, COCO nor ImageNet datasets didn't include any landforms(terrains). Besides, this pretrained model has 21 classes(we only have 6), which could cause imbalance. So we began cautiously with a few iterations(epochs). Once there was some promising results, we gradually increased the iterations to 1800, 4000 and 10000. Computation was another reason to proceed with this stepped approach to training. On average, each iteration takes about $4-6$ seconds depending on the host.

All the training and validation images had to be converted from the initial color map as given in the labelled image in Figure 7 to the new color map shown in Figure 5. Besides, the mapping between the colors and associated classes had to be corrected to align with the pretrained model. No weights were modified.

$Xception$ backbone is optimised for CPUs and that was the motivation to choose it. Initially, we tried to use $xception65\_cityscapes\_trainfine$(trained on Cityscapes Dataset [16]) checkpoint [15]. But, we ran into errors with it. So, we proceeded with $xception65\_coco\_voc\_trainaug$ checkpoint.

Even though it might seem simple to use $DeepLabv3+$, there are a lot of challenges associated with it.

- *Lack of documentation:* Even though $DeepLabv3+$ is part of tensorflow models [15], it's poorly documented. The installation instructions are outdated and it takes time to resolve dependencies.
- *Computation:* The model requires substantial computation. Our home or work PC isn't sufficient.
- *Configuration:* Since $DeepLabv3+$ doesn't have official documentation, it's difficult to configure it to suit our needs. It takes a lot of effort to fix issues and configure it.
- *Usage:* The input is not just the original and labelled images, it also requires conversion of labelled image into single-channeled gray scale images. We created a custom script [5] to do it.

D. Training Infrastructure

We used $DeepLabv3+$ [6] [15] [17] built on TensorFlow and Python for training. The models were executed on 2 separate machines - Machine 1 and Machine 2 on Chameleon Cloud [19]. The configuration of these machines are given below.

Machine 1	
OS	Ubuntu 18.04.5 LTS
TensorFlow	1.14.0
Python	3.6.9
Processor	2 × Intel(R) Xeon(R) Gold 6240R
Cores/threads	48/96
RAM	192 GB
Avg. training time/step	4 seconds

Machine 2	
OS	Ubuntu 18.04.5 LTS
TensorFlow	1.14.0
Python	3.6.9
Processor	2 × Intel(R) Xeon(R) CPU E5-2670 v3
Cores/threads	24/48
RAM	128 GB
Avg. training time/step	6 seconds

It takes approximately 11.25 - 16.75 hours to train a model for 10000 iterations.

TABLE I: mIOU of validation dataset

Class	**mIOU**
Terrain 1	71.44%
Terrain 2	21.27%
Terrain 3	54.46%
Terrain 4	39.55%
Terrain 5	94.1%
Background	93.61%
Average	62.41%

E. Results

Table I shows mean Intersection Over Union(mIOU) metric(accuracy) of the segmentation classes on validation data. Our validation contained just 9 images. It should be noted that this has pretrained model has a mIOU of 82.2% [15] on PASCAL VOC dataset [14]. After 10000 iterations, the cross-entropy loss was 0.2593. Figure 9 shows the trend of loss function as training progressed.

Figure 8 shows the comparison of an image, it's given label and predicted label by our trained model. For us, that prediction is fairly accurate. This is because, there are small labelling errors in our training set. From our experience of labelling terrains, sometimes we found difficult to distinguish terrains and clearly define a boundary.

1) Inferences:

- *Accuracy:* mIOU of the classes is proportional to the training data. There were only a few images for Terrains 2 and 4 when compared to the rest of the terrains.
- *Dataset has bias:* Majority of the dataset had images of Terrain 1, 5 and background. We should have had better judgement while preparing the dataset.
- *Labelling errors could have influenced training:* In Figure 8, the prediction is accurate than our label. It's a tedious task and sometimes we found it difficult to distinguish between terrains. Besides, these labels were converted from the initial labels created for Approach 2, which added noise.
- *Lack of fine tuning:* The model isn't fine tuned specifically to suit our needs. We haven't modified the wights to counter class imbalances which could have contributed to the results.

Fig. 8: Original image, our label and predicted label (from left to right)

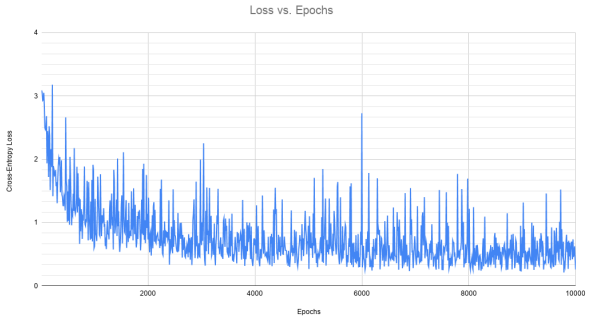


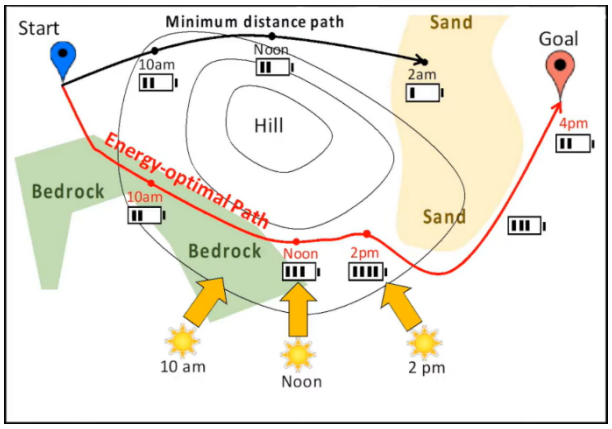
Fig. 9: Cross-entropy loss as training progressed

- *Need more training time:* Our model is trained for only 10000 iteration. From my understanding, most transfer training involving $DeepLabV3+$ has been trained for 30000 iterations. In our case, that's almost 34 - 50 hours of nonstop training, which was difficult.

Nevertheless, this results give us some insights. Besides, this can be considered as a successful experiment. In our case, the preferred terrain(Terrain 1) has over 70% accuracy and most unfavorable terrain(Terrain 5) has over 94% accuracy.

III. Navigation Algorithms

Having covered the methodologies for terrain classification both from satellite images (which we could not completely achieve) for global planning as well as from on-board camera images for local planning, the next step is to assign weight/cost functions to the terraneous area in which the rover moves from a source to a target, and then apply search algorithms based on heuristics to evaluate traversal costs calculated by them. The VeeGer project from NASA's Jet Propulsion Laboratory, uses similar methodology as shown in Figure 10. It also factors in availability of solar energy in cost considerations.



Source: VeeGer [23]

Fig. 10: Vision-based Estimation of Expending and Generating Energy for Rovers

Flat areas are associated with lower costs, while elevations are associated with higher costs. There are topographical images available from agencies like NASA [25] and ESA [26] which can be leveraged for the weight assignment to the image patches obtained as output from the DeepLab algorithms described above based on the image inputs it processes. For sandy dunes(as demonstrated in [27]), the weights associated to them may be calculated based on the relative time it takes to travel certain some distance on the sandy surface vs the hard surface. Since we could not complete the global planning part, we have created a random terraneous surface with assumed costs to show how the search

algorithms work in the attached python notebook. However, this should ideally be from the DeepLab outputs and assigned automatically as described.

There are multiple state space search algorithms like Dijkstra's algorithm, heuristic algorithms like A*-algorithm and incremental heuristic search algorithms like LPA* and D*-lite used for this purpose. Dijkstra's algorithm is more of an uninformed search since the target node is not known beforehand. However, in case of informed searches, heuristic algorithms like $A*$ are optimal and complete [28]. Optimal means that algorithms are sure to find the least cost from the source to the destination and Complete means that they are going to find all the paths that are available to us from the source to the destination.

A* algorithm forms the basis of the heuristic algorithm. We will discuss A* algorithm first and go over to its incremental search variants like LPA*, D* and D*-lite (which is more applicable to our use case). In the attached python notebook, we have shown functioning of both A* and D*-lite algorithms for calculating the path from source to destination over a terrain.

A. A* Algorithm

While traversing a path A* algorithm calculates the priority values(f) for each node it explores for reaching to the destination, using the cost value(g) it has travelled so far from the source to reach a specific point or state and a heuristic value(h) which is a lower estimate of the cost remaining to reach the destination. Mostly the heuristics are calculated based on the Euclidean distance, Manhattan distance or diagonal distance of the specific state to the destination state. Here we have used Euclidean distance. The pseudo-code for the algorithm [28] is shown in Figure 11.

But A*-algorithm is slow and the space it requires is higher, since it saves all the possible paths that are available to us. This leads to more efficient deliberative control of the rover while working with the global planning as discussed. The local planning needs dynamic adjustments to the cost functions, for which A* algorithm may not be efficient. We need incremental heuristic state space search algorithms as below. These algorithms reuse information from previous searches unlike A* which discards them.

```
let the openList equal empty list of nodes
let the closedList equal empty list of nodes
put the startNode on the openList (leave it's f at zero)
while the openList is not empty
    let the currentNode equal the node with the least f value
    remove the currentNode from the openList
    add the currentNode to the closedList
    if currentNode is the goal
        You've found the end!
    let the children of the currentNode equal the adjacent nodes
    for each child in the children
        if child is in the closedList
            continue to beginning of for loop
        child.g = currentNode.g + distance between child and current
        child.h = distance from child to end
        child.f = child.g + child.h
        if child.position is in the openList's nodes positions
            if the child.g is higher than the openList node's g
                continue to beginning of for loop
        add the child to the openList
```

Source: Edureka [28]

Fig. 11: A* Algorithm

Hence given a dynamic change in cost function in rover's observable space, A* needs re-computation from scratch. As noted, the priority functions are handled through *Priority Queue* data structure.

B. Lifelong Planning A* (LPA*)

An incremental version of A* which determines how to efficiently update a shortest path under changing edge costs when the start and end positions are static. The efficiency is achieved by only updating the values that need to be updated to find the shortest path [29].

C. D* Algorithm

D* algorithm resembles A* algorithm, except for the fact that the arc costs, may change during the traverse of the solution path. Provided that the traverse is properly coupled to the replanning process, it is guaranteed to be optimal. It was proposed by Anthony Stentz [30]. Like Dijkstra's and A* algorithms, D* too maintains an open list for the nodes to be iteratively evaluated, but works backwards from goal to start.

D. D*-lite algorithm

This appeared in 2002 AAAI paper [31] by Sven Koenig and Maxim Likhachev. D*-Lite is substantially light-weight than D*, since it uses only one tie-breaking criterion when comparing priorities, which simplifies the maintenance of the priorities, and does not need nested if-statements with complex conditions, which simplifies the analysis of the program flow. These properties also allow one to extend it easily, for example, to use inadmissible heuristics and different tie-breaking criteria to gain efficiency.

It is similar to LPA* except for the fact that here the start position is dynamic and equal to the present state [29] and cost function is calculated backwards from the target to the current position, in place from the start position, so g-values are actually the goal distances. The heuristic on the other hand is with respect to the source. This is also shown in the Python notebook attached. D*-lite shows significant performance improvements if there are lot of adjustments to the cost functions along the path. The pseudo-code for the algorithm [32] is shown in Figure 12.

```
CalculateKey(s)
    return [min(g(s), rhs(s))+h(sstart, s)+km; min(g(s), rhs(s))]
Initialize()
    U ← 0
    km ← 0
    for (s ∈ S) do rhs(s)← g(s)← ∞
    rhs(sgoal)← 0
    U.Insert(sgoal, CalculateKey(sgoal))
UpdateVertex(u)
    if (u <> sgoal) then rhs(u)← min s∈Succ(u)(c(u, s)+g(s))
    if (u □ U) then U.Remove(u)
    if (g(u) <> rhs(u)) then U.Insert(u, CalculateKey(u))
ComputeShortestPath()
    while (U.TopKey < CalculateKey(sstart) or rhs(sstart) <> g(sstart)) do
        kold ← U.TopKey()
        u ← U.Pop()
    if (kold < CalculateKey(u)) then
        U.Insert(u, CalculateKey(u))
    else
        if (g(u)>rhs(u)) then
            g(u)←rhs(u)
            for (s □ Pred(u)) do UpdateVertex(s)
        else
            g(u)← ∞
            for (s ∈ Pred(u) ∪ {u}) do UpdateVertex(s)

Main()
    slast = sstart
    Initialize()
    ComputeShortestPath()
    while (sstart <> sgoal) do
    // if (g(sstart)= ∞) then there is no known path
        sstart ← arg min s∈Succ(sstart)(c(sstart, s)+g(s))
        Move to sstart
        Scan the graph for changed edge costs
        if (any edge costs changed) then
            km = km+h(slast, sstart)
            slast = sstart
            for (all directed edges (u, v) with changed costs) do
                Update the edge cost c(u, v)
                UpdateVertex(u)
            ComputeShortestPath()
```

Source: David Mackay [32]

Fig. 12: D*-lite Algorithm

E. Implementation

This is indicated step by step in the attached python notebook. The output representation of the terraneous surface in 400x400 pixel bmp format is imported into the python notebook and OpenCV is used to convert the image into a 2x2 numpy matrix. Weights are computed based on the colors. In order to reduce computational costs, the matrix is divided into 80x80 grid with each window of 5x5 pixel to form a reduced matrix. Source and destination coordinates are entered and the respective A* and D* algorithms are able to compute the paths. The paths are also shown over the actual image. We have tried to insert weight variations in the path as well for the D*-lite algorithm to demonstrate its dynamic nature.

IV. Challenges

Every project has its own challenges and we thought that it's better to share it rather than hide it. We've already mentioned a few of them. However, we decided to highlight a few.

- *Labelled data:* When we began working on this project, we were confident that data wouldn't be an issue. But, it was not the case. Even though there was a lot of work done in autonomous vehicle navigation and related computer vision problems, there was absolutely no labelled data that suited our needs. This presented a huge challenge. It took considerable amount of time and effort to label the images.
- *Lack of exposure:* ML and DL domains are fairly new to us and we lacked insights. This was evident in the segmentation results. We didn't consider the ratio of classes while deciding on training data. Besides, fine tuning

models requires some insights in ML and DL.

- *Use of open source:* Open source is definitely beneficial. But, it brings its own problem. Lack of documentation haunted us. We spend considerable time in setting up $DeepLabV3+$.
- *Feasibility:* Initially, we thought that this project was manageable. This was under certain assumptions(data, previous related work, infrastructure etc). But, the moment we realised that our assumptions are wrong, the work load piled up. We were lucky that still things worked in our favor.

Nevertheless, we gained some valuable knowledge.

V. Conclusion

We consider this project as an exploration into AI technologies that we thought could be implemented. We have showed that entirely different AI domains can be combined to solve some complex problems. Rover maneuvering is a unique problem which has many constraints and it would be impossible to achieve it without these powerful solutions.

This was a great learning opportunity. When we learned $A*$ algorithm for state space search, we assumed that heuristics was available. In reality, obtaining the initial heuristics itself is a problem. Initially, we thought that rover required to perform only object classification for navigation. But, during the modelling of $A*$ algorithm, it was clear that we require semantic segmentation to classify the terrain.

We're aware that there are inherent assumptions in the modelling of the problem. But, it should be noted that the process of modelling begins with a simple problem.

VI. Future Scope

There are areas of improvement in this project. To begin with, we should definitely implement semantic segmentation for global planner. The $Xception$ model has to be fine tuned for class imbalances and our terrain specific features. $D*\ lite$ needs to be refined to improve path planning. Besides, the semantic segmentation has to be integrated with the navigation algorithms.

Future work could focus to develop a new model or to train this model without initial checkpoints. Current rovers doesn't have the computation capability to perform on-board learning. Instead, they use the trained models to classify terrain. We could attempt to develop a lite DL model suitable for on-board learning which would materialize the idea of *lifelong learning*. This would be very beneficial for space agencies because the rover could be repurposed for unplanned missions.

VII. Acknowledgement

We thank Dr. Pei Wang for his support. His words *"push the project as much as you can"* was the motivation when faced with challenges. We also express our gratitude towards Dr. Justin Shi, for his kind act of sharing his HPC infrastructure to run our experiments. Last but not least, we thank all the fellow netizens for their valuable suggestions and solutions. Without their help, this project wouldn't have materialized.

VIII. Artefacts

All the artefacts related to the project are shared on Google Drive.

- Navigation algorithms are in Google Drive/navigation_package
- Semantic segmentation package is in Google Drive/semantic_segmentation_package

Google Drive link in text: https://drive.google.com/drive/folders/183Ix3LEi-Kw7S5FyZ23agQ9IGdyegL0y?usp=sharing

References

[1] "MARS Reconnaissance Orbiter—HiRISE." Accessed May 1, 2021. https://mars.nasa.gov/mro/mission/instruments/hirise/.

[2] "HiRISE - Digital Terrain Models." Accessed May 1, 2021. https://www.uahirise.org/dtm/.

[3] "Mars Reconnaissance Orbiter." Accessed May 1, 2021. https://mars.nasa.gov/mro/.

[4] Rothrock, Brandon, Ryan Kennedy, Chris Cunningham, Jeremie Papon, Matthew Heverly, and Masahiro Ono. "SPOC: Deep Learning-Based Terrain Classification for Mars Rover Missions." In AIAA SPACE 2016. Long Beach, California: American Institute of Aeronautics and Astronautics, 2016. https://doi.org/10.2514/6.2016-5539.

[5] Sahu, Beeren. "How to Use DeepLab in TensorFlow for Object Segmentation Using Deep Learning." Medium, September 24, 2018. https://medium.com/free-code-camp/how-to-use-deeplab-in-tensorflow-for-object-segmentation-using-deep-learning-a5777290ab6b.

[6] Chen, Liang-Chieh, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation." ArXiv:1802.02611 [Cs], August 22, 2018. http://arxiv.org/abs/1802.02611.

[7] “Analyst’s Notebook.” Accessed May 1, 2021. https://an.rsl.wustl.edu/.
[8] “Mastcam - Instruments.” NASA’s Mars Exploration Program. Accessed May 1, 2021. https://mars.nasa.gov/msl/spacecraft/instruments/mastcam.
[9] “Home - Curiosity.” NASA’s Mars Exploration Program. Accessed May 1, 2021. https://mars.nasa.gov/msl/home.
[10] “Kaggle: Your Machine Learning and Data Science Community.” Accessed May 1, 2021. https://www.kaggle.com/.
[11] Wikipedia contributors, ”Adobe Photoshop,” Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Adobe_Photoshop&oldid=1020552344 (accessed May 4, 2021).
[12] “ImageNet.” Accessed May 2, 2021. https://image-net.org/.
[13] “COCO - Common Objects in Context.” Accessed May 2, 2021. https://cocodataset.org/home.
[14] “The PASCAL Visual Object Classes Homepage.” Accessed May 2, 2021. http://host.robots.ox.ac.uk/pascal/VOC/.
[15] GitHub. “Tensorflow/Models.” Accessed May 2, 2021. https://github.com/tensorflow/models.
[16] “Detailed Results – Cityscapes Dataset.” Accessed May 2, 2021. https://www.cityscapes-dataset.com/detailed-results/.
[17] Adam, Hartwig, Florian Schroff, George Papandreou, Yukun Zhu, and Liang-Chieh Chen. DeepLabV3+ (version commit 8c169ea4b7911854c986933126409c4757f579f1). TensorFlow Models, 2018.
[18] Rothrock, Brandon, Ryan Kennedy, Chris Cunningham, Jeremie Papon, Matthew Heverly, and Masahiro Ono. “SPOC: Deep Learning-Based Terrain Classification for Mars Rover Missions.” In AIAA SPACE 2016. Long Beach, California: American Institute of Aeronautics and Astronautics, 2016. https://doi.org/10.2514/6.2016-5539.
[19] “About Chameleon - Chameleon.” Accessed May 3, 2021. https://chameleoncloud.org/about/chameleon/.
[20] “Zooniverse.” Accessed May 5, 2021. https://www.zooniverse.org/projects/hiro-ono/ai4mars.
[21] ”Nasa-Jpl/Spoc_lite.” 2020. Reprint, NASA Jet Propulsion Laboratory, 2021. https://github.com/nasa-jpl/spoc_lite.
[22] Ono, Masahiro, Brandon Rothrock, Kyohei Otsu, Shoya Higa, Yumi Iwashita, Annie Didier, Tanvir Islam et al. ”MAARS: Machine learning-based Analytics for Automated Rover Systems.” In 2020 IEEE Aerospace Conference, pp. 1-17. IEEE, 2020.
[23] ”VeeGer: Vision-Based Estimation of Expending and Generating Energy for Rovers.” Accessed May 5, 2021. https://www.youtube.com/watch?v=0sGqkfV1q3k.
[24] “Cityscapes Dataset – Semantic Understanding of Urban Street Scenes.” Accessed May 5, 2021. https://www.cityscapes-dataset.com/.
[25] “Mars Terrain.” Accessed May 5, 2021. https://www.msss.com/http/ps/dtm/marsterrain.html.
[26] ESA. “The First Hiking Maps of Mars,” n.d. http://www.esa.int/Science_Exploration/Space_Science/Mars_Express/The_first_hiking_maps_of_Mars.
[27] Space Showcase. “Future Rovers May Walk Too - See A Prototype.” Accessed May 5, 2021. http://videos.space.com/m/AW2uzBeR/future-rovers-may-walk-too-see-a-prototype.
[28] Edureka. “A* Algorithm — Introduction to the A* Seach Algorithm,” December 4, 2019. https://www.edureka.co/blog/a-search-algorithm/.
[29] Stephens, Sam. “Understanding the Differences between LPA* and D* Lite,” n.d. https://github.com/samdjstephens/pydstarlite/wiki/Understanding-the-differences-between-LPA*-and-D*-Lite.
[30] Stentz, Anthony. The D* Algorithm for Real-Time Planning of Optimal Traverses. CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, 1994.
[31] Koenig, Sven, and Maxim Likhachev. ”D* lite.” Aaai/iaai 15 (2002).
[32] Mackay, David. ”Path planning with d*-lite.” DRDC Suffield TM 242 (2005).