Rover-IRL: Inverse Reinforcement Learning with Soft Value Iteration Networks for Planetary Rover Path Planning

Max Pflueger[1], Ali Agha[2], and Gaurav S. Sukhatme[1]

***Abstract*— Planetary rovers, such as those currently on Mars, face difficult path planning problems, both before landing during the mission planning stages as well as once on the ground. In this work we present a new approach to these planning problems based on inverse reinforcement learning (IRL) using deep convolutional networks and value iteration networks as important internal structures. Value iteration networks are an approximation of the value iteration (VI) algorithm implemented with convolutional neural networks to make VI fully differentiable. We propose a modification to the value iteration recurrence, referred to as the soft value iteration network (SVIN). SVIN is designed to produce more effective training gradients through the value iteration network. It relies on an internal *soft* policy model, where the policy is represented with a probability distribution over all possible actions, rather than a deterministic policy that returns only the best action. We demonstrate the effectiveness of our proposed architecture in both a grid world dataset as well as a highly realistic synthetic dataset generated from currently deployed rover mission planning tools and real Mars imagery.**

I. Introduction

Value iteration networks (VIN) are an approximation of the value iteration algorithm [1], [2] that were originally proposed by [3] as a way of incorporating a differentiable planning component into a reinforcement learning architecture. In this work, we apply VIN in an inverse reinforcement learning (IRL) approach for robot path planning problems. The architecture has two main neural network components, the VIN itself which is an unrolling of the value iteration recurrence to a fixed number of iterations, and the reward network which transforms terrain and goal data into a reward map that feeds into the VIN. An important feature of this approach is that the learned component of the network exists entirely within the reward network, the output of which must be a well behaved reward function for our planning problem, making human interpretation of the planning results relatively easy. In this work, we restrict ourselves to 2D path planning problems on grids that have only local neighbors. This is a natural constraint of using convolutional neural networks for implementing the value iteration algorithm, although one could imagine convolutional VINs of higher dimensionality.

Although the use of a reward function in value iteration is quite intuitive, writing down how to calculate a reward function to produce the desired planning results is deceptively difficult, as has been observed by researchers in the past [4]. Part of the difficulty comes from the need to keep the relative costs and rewards of different types of terrain in balance with each other. Systems that learn a reward function can side step these manual difficulties by requiring the reward function to perform well and allow the relative weights of different risks to come into balance automatically. Figure 1 shows an instance of one of these learned reward functions from our dataset and some resulting path plans.

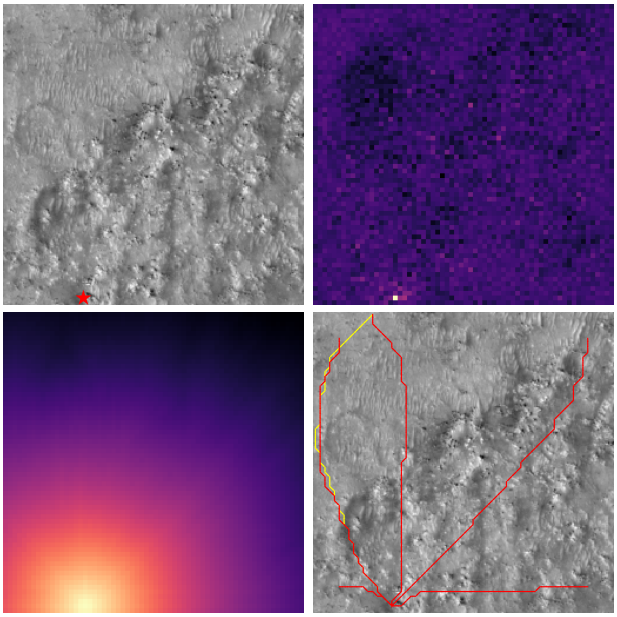

Fig. 1. Test example from our Jezero Crater dataset (best viewed in color). **Top Left:** Satellite map shown in grey scale, this image covers a 64m x 64m square of terrain with a pixel size of 25cm per pixel. The red star shows the goal position. **Top Right:** Learned reward function: Darker colors show low reward areas and brighter colors show high reward areas. Note the single bright pixel at the goal position. **Bottom Left:** Value function: shows calculated expected discounted reward when starting from each location. Dark to light shows low to high reward values. **Bottom Right:** Satellite map with paths: Example paths from our policy (as implied by the value function) are shown in red, the demonstration path from the dataset is shown in yellow (rendered under the red paths). Red paths are shown starting from 4 corners of the map as well as the start of the yellow demo path. In this example we see one plan trajectory that closely follows the demo, and another that takes an alternate path around an undesirable region of terrain. The region being avoided shows a ripple pattern that is a common feature in this dataset that we see being avoided by both the expert planner and our learned trajectories.

A. Planetary Rover Path Planning

Planetary rovers, such as those currently being operated on Mars, face difficult navigation problems in both short and long range planning [5]. These problems are made more difficult

Max Pflueger would like to acknowledge support from the ARCS Foundation for this work.

[1] Max Pflueger and Gaurav Sukhatme are in the Department of Computer Science at the University of Southern California, Los Angeles, CA 90089 USA. {pflueger,gaurav}@usc.edu

[2] Ali Agha is with the Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109 USA. aliakbar.aghamohammadi@jpl.nasa.gov

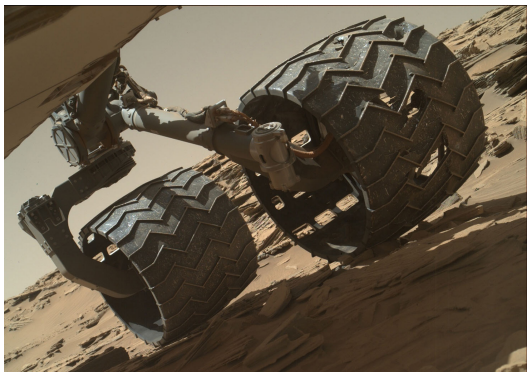

Fig. 2. MSL image showing damage to the wheels from small rocks. The rocks that cause this damage are too small to see in satellite images, but may co-occur with certain mineral types and terrain features that can be identified from orbit.

by tight bandwidth constraints and time lag to communicate with Earth that operationally limit the rovers to only making one round trip communication with Earth per sol (a sol is a Martian day, about 24 hours 40 minutes). Existing rover driving techniques rely mostly on imagery taken on the surface to plan actions, however those plans can only go as far as the rover can see.

Rovers on the ground are naturally limited in their ability to see by variations in terrain and obstructions, as well as by limitations of the cameras. Orbital imagery can be provided at effectively unlimited distance, but arrives at lower resolution than images on the ground. Although orbital imagery can be processed to obtain certain characteristics of the terrain, such as coarse elevation, or in some cases estimates of terrain type, relating these characteristics to a cost function relevant to navigation is nontrivial. In particular, many dangers to the Mars Science Laboratory (MSL) “Curiosity" rover are too small to be seen directly in orbital imagery (Figure 2), but it might be possible to find terrain or mineral patterns that are associated with dangerous surface features.

We observe that this problem possesses two important properties: we can formulate useful algorithms for long range planning if a suitable cost function is available, and short range planning techniques exist (based on surface imagery) that are both sophisticated and reliable.

In this work, we present an inverse reinforcement learning architecture using a variant on value iteration networks we call *soft* value iteration networks. This system implicitly learns to solve navigation planning problems using training data derived from other planners or historical rover driving data. These navigation functions only depend on orbital data to produce a plan, and are thus useful at ranges beyond what is visible from the ground. Because the inverse reinforcement learning architecture we use is structured around value iteration, it implicitly produces useful cost functions for navigation. We propose an architecture that allows this planning product to be integrated with existing human expert-designed local planners to maintain guarantees in safety critical systems.

An additional application of our technique is that as a data driven planning technique based only on information obtained from orbit, our planner could be used in mission planning scenarios for tasks such as evaluating the value of a landing site. Planners that are currently used for this purpose rely on heuristic rules for travel cost and maps created through a terrain labeling process.

II. Related Work

Deep reinforcement learning (RL) techniques have been frequently applied in settings where we wish to model an action policy as a function of robot state for both discrete and continuous action spaces [6], [7], [8]. Reinforcement learning has also been used in challenging space applications for hopping rovers [9] using least-squares policy iteration [10].

Value iteration networks were first explored by [3] in the context of reinforcement learning. They developed the value iteration module as an approximation of the value iteration algorithm that could be used inside a reinforcement learning algorithm to allow the algorithm to ‘learn to plan’. We demonstrate a different application of the value iteration module from their work: instead of allowing it to have abstract spatial meaning with a learned state transition model and pass through a final policy network, we bind it tightly to the map of our terrain and the available state transitions. By doing this our algorithm is explicitly forced to learn the reward function that produced the observed behavior in our IRL formulation of the problem. This is important to us as it allows the reward map to be interpretable in other contexts. QMDP-net, proposed by [11], uses a planning technique similar to VINs, but adapted for partially observable problem settings. Other researchers have also looked at ways of using neural networks to model dynamic programming and planning techniques [12], [13].

IRL for navigation has been studied by other groups as well [14], [15], [16], for example the LEARCH technique proposed by [17] has a very similar functional input/output design, although it relies on substantially different computational tools. [16] and [18] combined the MaxEntIRL framework with a deep network similar to our own for computing cost maps. Imitation learning has also been done with deep neural network (DNN) based policies (instead of cost functions) and visual state information [19].

Our formulation of IRL with VINs shares a substantial mathematical similarity with previous work in maximum entropy inverse reinforcement learning [15], [16]. In particular, our use of a ’soft’ action model (discussed in section IV-E) in combination with a softmax cross-entropy loss function makes our optimization objective very similar to the MaxEntIRL objective. The main difference comes from how the probabilistic action model in the maximum entropy approach is normalized over full trajectories, where as our soft action model is normalized over local actions. We will discuss this comparison in more detail in subsection IV-F.

III. Preliminaries

We use the problem formulation of a Markov Decision Process $M = (S, A, P, R, \rho)$ where $\rho(s)$ defines the distri-

bution over initial states, and a robot in state $s \in S$ must choose an action $a \in A$, which will give reward $r = R(s, a)$ and proceed to state s' with probability $P(s'|s, a)$. We say that a policy $\pi(a|s)$ will define a policy distribution over actions conditioned on states, and that $\tau^\pi(s_0, a_0, s_1, a_1, ...)$ is the distribution over trajectories created recursively when $s_0 \sim \rho(s)$, $a_t \sim \pi(a|s_t)$, and $s_{t+1} \sim P(s_{t+1}|s_t, a_t)$.

We then define the value function V conditioned on policy π as

$$V^\pi(s) = \mathop{\mathbb{E}}_{s,a\sim\tau}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t, a_t)\right] \tag{1}$$

for some discount factor $\gamma \in [0, 1]$. Here the notation $\mathbb{E}_{s,a\sim\tau}$ refers to the expected value of the expression with s and a drawn from the distribution τ. This can be rewritten recursively as

$$V^\pi(s) = \mathop{\mathbb{E}}_{a\sim\pi(a|s)}\left[R(s, a) + \gamma\sum_{s'} P(s'|s, a)V^\pi(s')\right] \tag{2}$$

We further define Q^π and rewrite the value function:

$$Q^\pi(s, a) = R(s, a) + \gamma\sum_{s'} P(s'|s, a)V^\pi(s') \tag{3}$$

$$V^\pi(s) = \mathop{\mathbb{E}}_{a\sim\pi(a|s)}[Q^\pi(s, a)] \tag{4}$$

We define the optimal value function $V^*(s) = \max_\pi V^\pi(s)$ and a policy π^* as an optimal policy if $V^{\pi^*} = V^*$. The value iteration algorithm can be used to find V^* through the following iterative process:

$$Q_n(s, a) = R(s, a) + \gamma\sum_{s'} P(s'|s, a)V_n(s') \tag{5}$$

$$V_{n+1}(s) = \max_a Q_n(s, a) \tag{6}$$

It is well known that as $n \to \infty$, $V_n \to V^*$, and an optimal policy π can be inferred as $\pi^*(a|s) = 1_{\mathrm{argmax}_a\, Q_\infty(s,a)}(a)$.

IV. Algorithm

We develop an inverse RL problem formulation based on using Value Iteration Networks (VIN) to backpropagate gradients through the planning algorithm to update the reward function. By contrast to previous work using VINs ([3]), we require the value iteration module to operate directly on the state space of our planning problem (rather than an inferred state space as in [3]). Although this restricts what can be done in the learning process (the problem must have a 2D grid state space, and linear state transition functions), by forcing a traditional interpretation on the value map we can use it with any control policy that is designed to incorporate this kind of data. In particular this includes the expert designed local control algorithms that provide the safety guarantee necessary for operating high value planetary rovers.

A. Architecture

The data flow of our algorithm is shown in Figure 3. We start by stacking visual map data along with a one-hot goal map and any other relevant data layers (such as elevation). These pass through the network f_R to produce the reward map that feeds into the value iteration module. Also feeding into the value iteration module are the state transition and reward kernels, denoted here as f_P.

The output from the VI module is used differently depending on whether we are currently training the algorithm or deploying it on an operational system. During deployment the value map from the VI module can be fed to an expert designed local planner and used in combination with high resolution local information to make planning decisions for a long range goal. During training the output from the VI module is compared with actions from a plan demonstration (which may come either from an expert planning algorithm, or from historical operations on a real platform), and the difference is used to calculate a loss function.

B. Training

In general this architecture can have trainable parameters in both f_R and f_P, however in our problem we fix the transition and reward kernels f_P and only train parameters in the network f_R. During training we attempt to minimize over θ the loss function defined below as:

$$L_\theta = -\sum_{s\in S_y}\sum_{a\in A} y(s, a)\log\frac{\exp Q_\theta(s, a)}{\sum_{i\in A}\exp Q_\theta(s, i)} \tag{7}$$

where S_y is the set of states on the training path and y is an indicator function for the action taken at each state on the training path. This may be recognized as a common softmax cross-entropy loss function, implying a probabilistic action policy based on the softmax of Q_θ over actions.

C. Plan Execution

The network is provided (as input) an orbital image of the relevant terrain, a goal position, and optionally other data products such as stereo elevation, surface roughness estimates, or hyper-spectral imagery. With a forward pass through the network we calculate the value and Q function estimates for that terrain and goal position. The policy implied by the Q function could be used directly, such as in a mission planning application, or the value and/or Q functions could be used in any other application where such estimates would be useful.

An important example is using the estimated value function as a tool for long range planning in combination with a more accurate or safe short range planner. A rover on the ground has more accurate data for short range planning within its radius of sight than is available from orbit. Within the radius of its sight, the rover is able to use expert algorithms for navigation, but can combine calculated path costs with value estimates at its planning horizon to choose an appropriate plan that considers longer range objectives. Replanning can then happen as frequently as is allowed by on board resources.

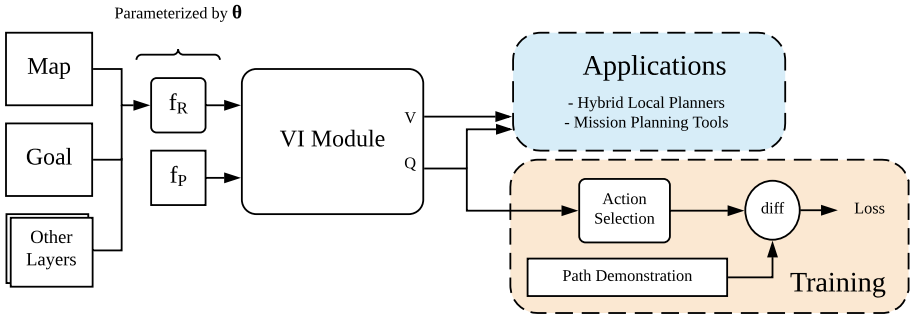

Fig. 3. Information flow diagram for our architecture. Map data may consist of visual overhead images, as well as other data products such as hyperspectral image channels, or elevation data. This is stacked with a one-hot representation of the goal to form a multi-channel image passed to the reward network f_R. The reward network is parameterized by θ, and its exact architecture may vary with the problem domain, our architecture for the JEZ dataset is shown in Figure 10. The VI Module receives the reward map as well as the state transition model f_P, and calculates the value map V and Q-function. During training, an action selection process infers the policy from the Q-function and then compares that with the demonstrated path to get a loss value. Applications may take advantage of V or Q, depending on their needs.

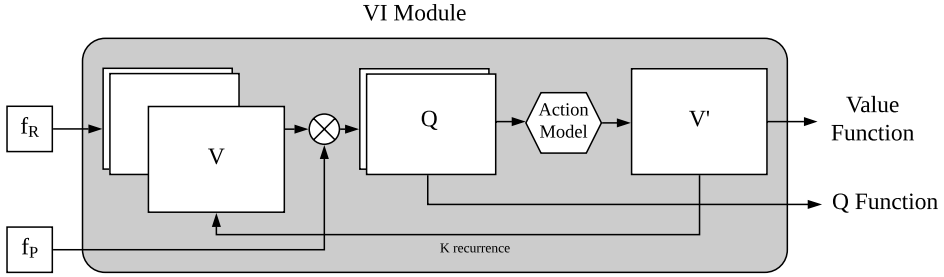

Fig. 4. Value Iteration Module. The (potentially multi-channel) reward map f_R is stacked with the current estimate of the value function V (typically initialized with 0's). The state transition model f_P is delivered as the convolutional kernel that creates the next estimate of Q. Q has as many channels as available actions. The action model then processes Q into the next estimate of V. Traditionally this is a max pooling operation, however in subsection IV-E we discuss our proposed soft action model. This process repeats for K iterations before outputting the final estimates of V and Q.

D. Value Iteration Module

The value iteration module uses the value iteration iterative process defined in Eqs. 5 & 6 to perform an approximation of value iteration for a fixed number of iterations k. The approximate nature of this module derives from the need to choose the fixed number of iterations k a priori, instead of iterating to convergence as required by the traditional value iteration algorithm. A representation of the architecture of the value iteration module is shown in Figure 4.

The two inputs f_R and f_P provide the reward map and convolutional kernel respectively. The reward map is stacked with the value map from the previous iteration and then convolved with f_P to produce a map of Q values. The Q channels must then be collapsed into the next value map through a model of the action selection process. Strict adherence to the value iteration algorithm requires that this be a max pooling operation (an optimal policy chooses the action of maximum reward), however, in the next section we propose an alternative approach.

E. Soft Action Policy

The traditional formulation of the value iteration algorithm requires that updates be done using the optimal action policy of always choosing the action of highest Q value as in Eq. 6 above. This is a theoretically well justified choice for a planning algorithm. However, in value iteration networks we have an additional objective to provide an effective gradient through the algorithm. If we assume a reward function R_θ parameterized by θ, we can calculate the gradient of the value function with respect to θ after k iterations as:

$$\nabla_\theta V_k(s) = \nabla_\theta R_\theta(s, a^*) + \gamma \sum_{s'} P(s'|s, a^*) \nabla_\theta V_{k-1}(s') \tag{8}$$

where, a^* is the optimal action selected by the max Q value in iteration $k-1$. Assuming a deterministic state transition model P (as is the case for the problems studied in this paper), this equation can be further simplified and expanded as:

$$\begin{aligned} \nabla_\theta V_k(s) &= \nabla_\theta R_\theta(s, a^*) + \gamma \nabla_\theta V_{k-1}(s') \\ &= \nabla_\theta R_\theta(s, a^*) + \gamma \nabla_\theta R_\theta(s', a'^*) + \gamma^2 \nabla_\theta V_{k-2}(s'') \end{aligned} \tag{9}$$

The key observation from Eq. 9 is that the gradient through the value function will only incorporate information about states that are reached by the best actions under the current reward function. In this work, we propose a modification to the value iteration algorithm that leads to more effective gradient values, which enhances the network training, particularly in the early training stages. Instead of using the value update from Eq. 6, we propose the following:

$$\begin{aligned} V_{n+1}(s) &= \sum_a Q_n(s, a) \frac{\exp(Q_n(s, a))}{\sum_{i \in A} \exp(Q_n(s, i))} \\ &= \mathop{\mathbb{E}}_{a \sim \pi_n} [Q_n(s, a)] \end{aligned} \tag{10}$$

$$\pi_n(a|s) = \frac{\exp(Q_n(s,a))}{\sum_{i \in A} \exp(Q_n(s,i))} \tag{11}$$

This formulation can be interpreted as performing value iteration under the assumption of a probabilistic action policy π_n rather than an optimal action policy (in this MDP formulation, the optimal action policy is the argmax over actions of the Q function, which is always deterministic). In the traditional formulation with deterministic state transitions, the reward gradient cannot carry information about action selections in sub-optimal successor states. However, we can see that if we attempt to take the gradient of Eq. 10 w.r.t. θ, we will not be able to remove the sum over actions and corresponding successor states. As such all possible future states and actions will (recursively) participate in the gradient. We speculate that this action model may also be beneficial for problems with probabilistic state transitions, although we suspect the benefits would be less pronounced.

F. Importance of Soft Policies in IRL

Inverse reinforcement learning in general has to deal with a problem that optimal policies are also deterministic, as we see in the form of traditional value iteration. This is problematic for IRL algorithms as it makes it difficult to calculate useful gradients that can be used to update a parameterized reward function. Our method and previous work (such as maxEnt IRL [15], [16]) aim at solving this problem by softnening the policies. Specifically, methods that rely on the principle of maximum entropy tend to make the policy class probabilistic, which then leads to a derivation of a practical gradient calculation [15], [16]. These techniques, known as MaxEnt IRL, define their policy as proportional to the exponential of the expected discounted reward of full trajectories.

In our work we address this challenge from a different perspective: Backpropagation through a VIN allows us to calculate gradients of the deterministic optimal policy. As discussed in previous sections, when those gradients are not useful, we modify our policy class to be probabilistic by making it proportional to the exponential of expected discounted reward. In our approach, however, this stochasticity is added over local actions. This new approach results in subtly different normalization across trajectories. In our system, trajectories of similar reward can receive different probability as a result of the graph structure. In other words, while the MaxEnt objective attempts to maximize the probability of the "demonstrated trajectories", our model attempts to maximize the probability of the "individual actions" in the trajectory demonstration.

While all these soft policy strategies attempt to converge to the same point (i.e., to a deterministic policy that matches the demonstration), they have different reward contours around that optimal point. The SVIN work proposed here provides a new mathematical counterpoint for developing useful IRL gradients, paving the way for new options for future development.

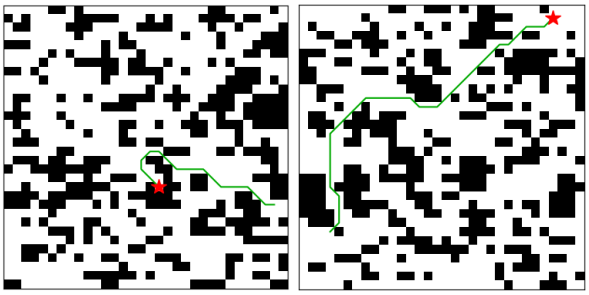

Fig. 5. Two example maps and paths from our grid world dataset. The star shows the goal position with one example path to that goal. The black cells are considered impassible and example paths are calculated with a standard search based planner.

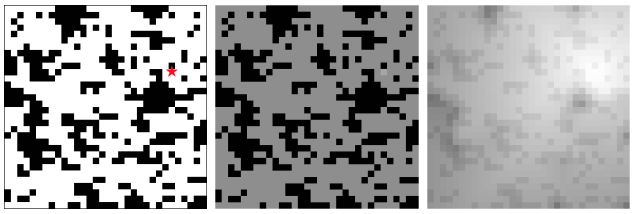

Fig. 6. **Left:** An example grid world obstacle map, the red star denotes the goal position. **Middle:** The corresponding reward map after training. Reward values are shown in grey scale with dark values being low reward. Note the slightly brighter goal position. **Right:** Value map produced by the SVIN algorithm. Grey values here show the expected discounted reward for all positions on the map. We see the shapes of the obstacles from the map, as well as 'shadows' cast by the obstacles across regions of the state space made less accessible (from the goal) by their presence.

V. Experiments

In this section, we demonstrate the performance of the proposed inverse reinforcement learning framework. We are testing our algorithm with two datasets. The first is a simplistic grid world dataset designed to show that the learning objectives and formulation of our algorithm are realistic. The second is a synthetic set of paths produced with actual Mars terrain data from expert terrain traversability labelings. All our datasets use a grid state space and a deterministic action model with 9 available actions corresponding to 4 straight neighbors, 4 diagonal neighbors, and 1 action to stay in place.

We track the performance of our models using two metrics, loss and accuracy. Loss refers to the standard softmax cross-entropy loss function described earlier in the paper. Loss is the function that is being optimized via gradient descent. Accuracy refers to the fraction of steps along the demonstrated path where optimal actions predicted by our network match the actions selected by the path data. The accuracy metric is not differentiable and hence cannot be optimized directly, however it is a better proxy for the useful performance of the network than the loss metric. Therefore, we primarily track progress in the accuracy value during training.

A. Grid World

The grid world dataset was designed to test the behavior of our algorithm in a simplistic environment that still required non-trivial planning behavior with significant look-ahead in the map.

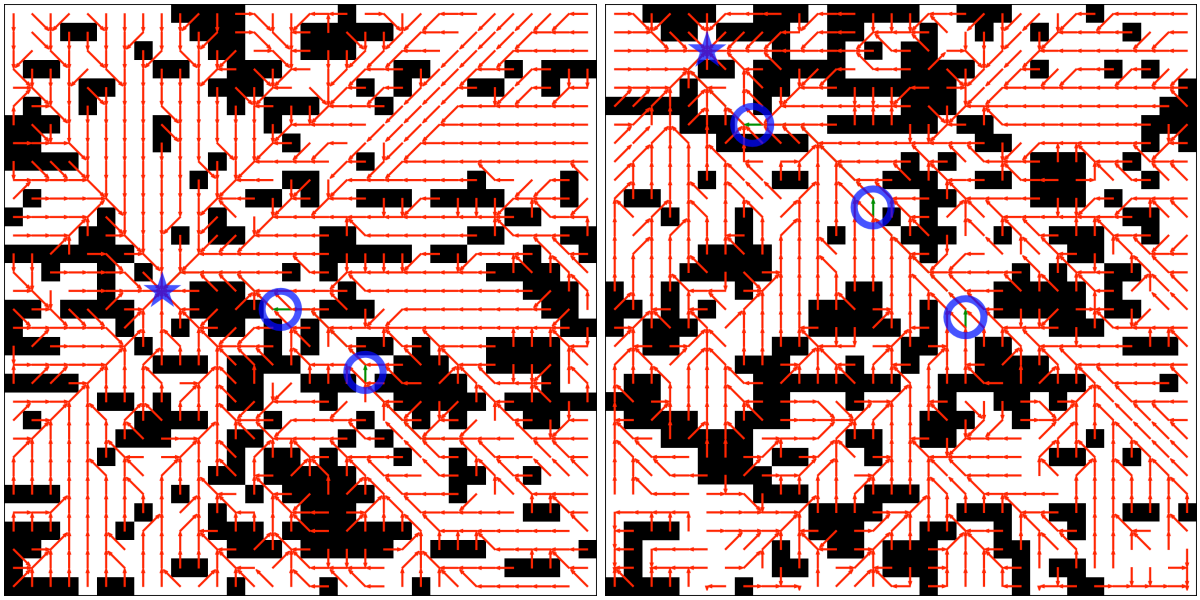

Fig. 7. Vector fields show optimal policy planning results on our grid world dataset. Learned policy actions are shown in red and the goal is denoted by a blue star. A few green arrows (highlighted in the blue circles) show places where the demonstrated actions from the training data deviated from our learned (red) policy. On the left we see a case where our algorithm did not perfectly predict the demonstration because of an identical length path that it took instead. In the map on the right, we can see a different form of error where some regions do not show valid policies that converge to the goal (bottom-right and bottom-left corners). This is an artifact of fixing the number of iterations (i.e., k) in VI. When k is too small, information about the goal location cannot propagate to the whole map.

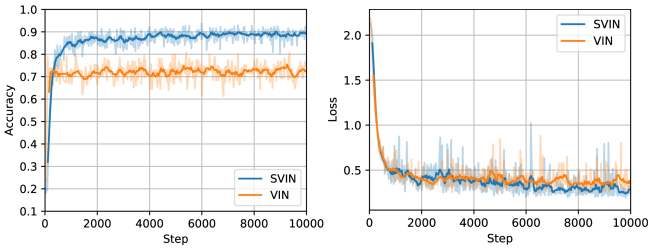

Fig. 8. Training curves show accuracy and loss on the test set against gradient steps on the grid world dataset. The model in orange uses a standard hard action model, the blue uses our proposed soft action model. Solid lines show a moving average of the shaded raw performance. Loss is described in Eq. 7. Accuracy is calculated as the percentage of grid cells in which the calculated optimal action and the action shown in the demonstration match. Though the accuracy value is more intuitive and easier to track, it is not differentiable like the loss value which is the optimization objective during training. Note the substantial improvement of our SVIN algorithm over standard VIN in accuracy.

1) Details: Each map has resolution of 32x32 cells. For each map we randomly choose start and goal positions and compute the shortest path. The full dataset consists of 1000 maps with approximately 50 paths per map. (The number of paths per map is approximate because occasionally a start and goal position are chosen for which there is no valid path in the map.) We split the grid world data by maps into 90% training and 10% test. Figure 5 shows a couple of sample paths on their corresponding maps from the grid world dataset.

The reward network f_R we use for grid world consists of 2 convolutional layers with 1x1 kernels (spatial context is not important for this problem). The first has 16 output channels and relu nonlinearity; the second has a single linear output channel.

2) Results: Figure 8 shows our training curves on the grid world dataset. While the standard VIN algorithm with a hard action model plateaus around 70-75% accuracy, our SVIN algorithm is able to reach a substantially higher level of performance on this dataset (89%).

It is also worth noting that the optimal upper performance bound is likely less than 100% and thus our actual performance is more than 89% of the optimal solution. There are two reasons for this: First, the grid world environment often contains multiple paths of identical length, and, structurally, the network is not capable of learning tie-breaking preferences. Additionally, the network is constrained by the choice of hyper-parameters. In particular, k controls how far information can propagate through the network. Our grid world network is trained with $k = 64$, which is longer than most paths, but under some conditions still may not be enough to converge rewards across the whole map.

Figure 7 illustrates the performance of our grid world network, as well as potential failure modes. The learned policy is shown in red. A few green arrows show where the training path deviates from the learned policy. In both maps the goal position can be identified as the major point of convergence of the vector field. The left map shows some instances of deviations of identical length. The right map has some distant areas of the map (bottom-right and bottom-left corners) that show clearly incorrect behavior.

Figure 6 visualizes the behavior of a trained network. It shows the input map on the left, the result after passing through the reward network in the middle, and the output value map on the right. It can be seen that SVIN correctly identifies the goal position as the brightest square in the reward map (middle graph). Also, in the value map (right graph), we see how the obstacles appear to cast shadows

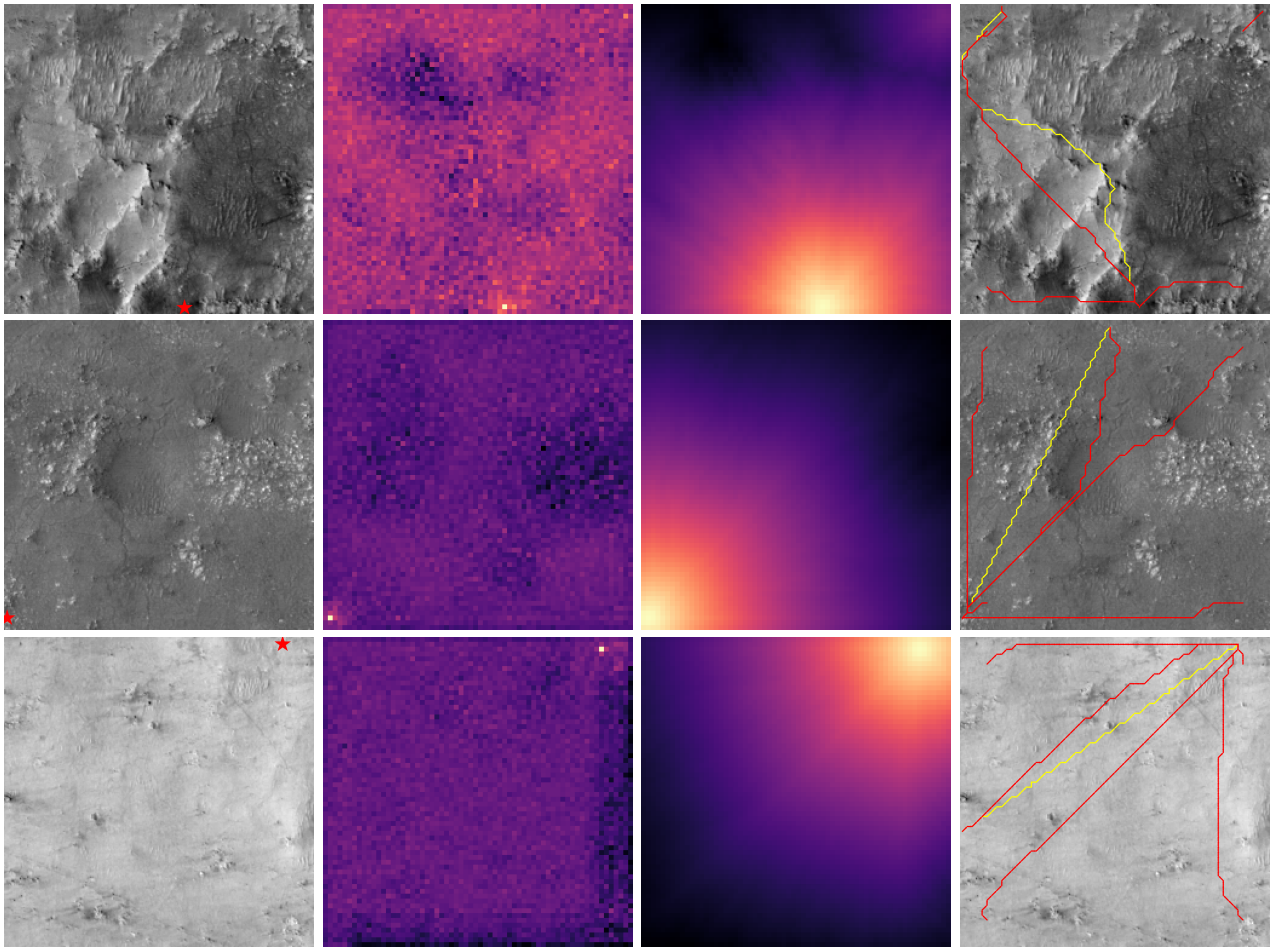

Fig. 9. Additional test examples from our Jezero Crater dataset, rendered in the same style as Figure 1. Left to right: satellite map, reward function, value function, satellite map with paths, example paths from our policy are shown in red, the demonstration path from the dataset is shown in yellow (rendered under the red paths). Row 1: Note how the trajectory instance started with the yellow demo path follows it closely at first, then deviates through a region the demo avoided. The upper right path has started in an area of the map that did not receive sufficient reward signal from the goal, and is thus showing undesirable behavior. Row 2 & 3: Here we see instances were the demo trajectory is simply a straight line, probably because the terrain labels were too coarse to pick out any features here. Nonetheless, our learned model had identified some meaningful terrain variations. Particularly in row 2 we see it steering through a smoother area to avoid a group of rocks.

through the space.

B. Jezero Crater

Jezero crater is a candidate landing site for the Mars 2020 rover, and as a part of the landing site evaluation process detailed maps have been created of the area along with a traversiblity tool that can produce minimum time paths between arbitrary waypoints [20]. These traversibility calculations are based on heuristics written by expert rover drivers as a function of satellite-based terrain classification, rock abundance, and slope. We used this traversibility tool to generate a dataset of 9010 terrain tiles (each 64m square) containing a path from this tool. Approximately 10% of the dataset is held out as a test set.

A diagram of the reward network we used with this dataset is shown in Figure 10. Table I lists some important training parameters. Training on this dataset, our model approaches 75% accuracy, as shown in Figure 11. As in the grid world example, these accuracy numbers can be

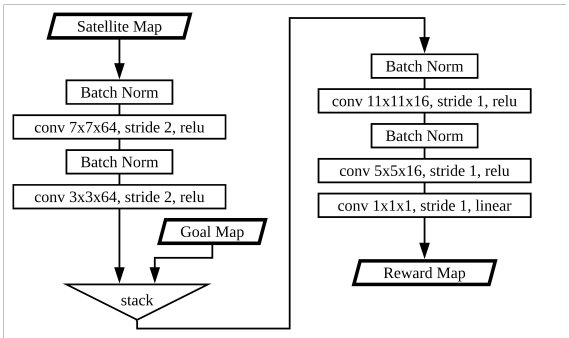

Fig. 10. Architecture of the reward network trained for the Jezero Crater dataset.

difficult to interpret, but to get a qualitative feel for what that 75% looks like, we have shown a representative set of examples of what the paths generated by our policy look like in Figure 1 and Figure 9. Although a probabilistic action model is used by SVIN for calculating the value function, the paths shown here are generated using a best action policy, so at test time the trajectories are not probabilistic. These

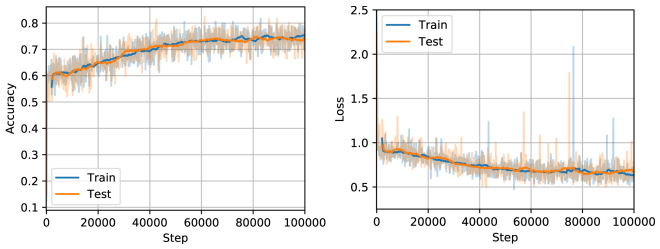

Fig. 11. Accuracy and loss training curves on the Jezero Crater dataset. Solid lines show a moving average of the shaded raw performance. Loss and accuracy are calculated here in the same way as the grid world dataset shown above.

figures can be compared with Figure 7, except in this case we have only rendered a collection of paths instead of the full vector field for visual clarity. A qualitative analysis of these results suggests that most of the time our network is generating policies that are goal directed and make an effort to avoid unsafe looking terrain. In some cases we have even seen instances where the policy from our network seems to outperform the training data (possibly due to relatively coarse terrain label data), an observation that will receive more scrutiny in future work.

Training Parameters	
γ (discount factor)	0.98
k (VIN Iterations)	150
Optimizer	Adam
Learning Rate	0.0001
Batch Size	20

TABLE I
JEZERO CRATER TRAINING PARAMETERS

An important qualitative observation is that even when the shown paths are nearly identical, there can still be a substantial fraction of states where our policy makes a different decision from the demonstration trajectory. This goes to suggest that the accuracy metric tracked during training can likely not be pushed all they way to 100%, and even the approximately 75% achieved by the model shown here may be close to the upper limit. A valuable direction for future work may be to develop a less noisy loss metric for training, that better represents policy similarity.

VI. Discussion and Future Work

In this paper we have demonstrated an inverse reinforcement learning architecture using soft value iteration networks (SVIN) that can be applied to path planning problems with planetary rovers. We have analyzed how the SVIN formulation can improve training gradients for problems with deterministic state transition dynamics, and have seen that improvement empirically on our gridworld dataset. We also applied our technique to a new synthetic set of paths on real Mars terrain, and seen how we can substantially approximate the performance of the expert planner used to generate the data, with some hints that it may even be possible to outperform the training data.

As we move forward with this project we plan to look into the viability of using reward networks trained on short paths to produce policies on much larger maps, a capability which would be key for mission planning and long range navigation applications.

VII. Acknowledgements

The authors would like to acknowledge the contributions of Bradd Carey and Sammi Lei in assisting with data preparation. Thank you to Kyohei Otsu and his team for access to the Mars Terrain Taversibility Tool.

References

[1] R. Bellman, *Dynamic programming*. Princeton, NJ: Princeton University Press, 1957.
[2] D. P. Bertsekas, *Dynamic programming and optimal control*. Athena scientific Belmont, MA, 1995.
[3] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 2154–2162.
[4] J. A. Bagnell, D. Bradley, D. Silver, B. Sofman, and A. Stentz, "Learning for autonomous navigation," *IEEE Robotics & Automation Magazine*, vol. 17, no. 2, pp. 74–84, 2010.
[5] D. Gaines, R. Anderson, G. Doran, W. Huffman, H. Justice, R. Mackey, G. Rabideau, A. Vasavada, V. Verma, T. Estlin, *et al.*, "Productivity challenges for mars rover operations," in *Proceedings of 4th Workshop on Planning and Robotics (PlanRob)*. London, UK, 2016, pp. 115–125.
[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[7] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.
[8] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.
[9] B. Hockman and M. Pavone, "Stochastic motion planning for hopping rovers on small solar system bodies," in *Proceedings of ISRR*, 2017.
[10] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *Journal of machine learning research*, vol. 4, no. Dec, pp. 1107–1149, 2003.
[11] P. Karkus, D. Hsu, and W. S. Lee, "Qmdp-net: Deep learning for planning under partial observability," in *Advances in Neural Information Processing Systems*, 2017, pp. 4697–4707.
[12] R. Ilin, R. Kozma, and P. J. Werbos, "Efficient learning in cellular simultaneous recurrent neural networks-the case of maze navigation problem," in *Approximate Dynamic Programming and Reinforcement Learning, 2007. ADPRL 2007. IEEE International Symposium on*. IEEE, 2007, pp. 324–329.
[13] D. Silver, H. van Hasselt, M. Hessel, T. Schaul, A. Guez, T. Harley, G. Dulac-Arnold, D. Reichert, N. Rabinowitz, A. Barreto, *et al.*, "The predictron: End-to-end learning and planning," *arXiv preprint arXiv:1612.08810*, 2017.
[14] D. Silver, J. A. Bagnell, and A. Stentz, "Learning from demonstration for autonomous navigation in complex unstructured terrain," *The International Journal of Robotics Research*, vol. 29, no. 12, pp. 1565–1592, 2010.
[15] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning." in *AAAI*, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.
[16] M. Wulfmeier, D. Z. Wang, and I. Posner, "Watch this: Scalable cost-function learning for path planning in urban environments," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 2089–2095.
[17] N. D. Ratliff, D. Silver, and J. A. Bagnell, "Learning to search: Functional gradient techniques for imitation learning," *Autonomous Robots*, vol. 27, no. 1, pp. 25–53, 2009.
[18] M. Wulfmeier, P. Ondruska, and I. Posner, "Maximum entropy deep inverse reinforcement learning," *arXiv preprint arXiv:1507.04888*, 2015.
[19] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, *et al.*, "A machine learning approach to visual perception of forest trails for mobile robots," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 661–667, 2016.
[20] M. Ono, B. Rothrock, E. Almeida, A. Ansar, R. Otero, A. Huertas, and M. Heverly, "Data-driven surface traversability analysis for Mars 2020 landing site selection," in *IEEE Aerospace Conference*, 2016.