Learn and Link: A Learning-Amenable Planning Paradigm

Daniel Molina, Kislay Kumar, Siddharth Srivastava
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University
Tempe, Arizona 85281

***Abstract*— In this paper, we present a new approach to learning for motion planning (MP) where *critical regions* of an environment with low probability measure are learned from a given set of motion plans and used to improve performance on new problem instances. We show that convolutional neural networks (CNNs) can be used to identify critical regions for motion planning problems.**

We also introduce a new suite of sampling-based motion planners, *Learn and Link*. Our planners leverage critical region locations identified by our CNN to overcome the limitations of uniform sampling, while still maintaining guarantees of correctness inherent to sampling-based algorithms. We evaluate Learn and Link against planners from the Open Motion Planning Library (OMPL) using an extensive suite of experiments on challenging motion planning problems. We show that our approach requires far less planning time than existing sampling-based planners.

I. Introduction

The motion planning (MP) problem deals with finding a feasible trajectory that takes a robot from a start configuration to a goal configuration without colliding with obstacles. From a computational complexity point of view, even a simple form of the MP problem is NP-hard [1]. In order to achieve computational efficiency, motion planning methods relax requirements of completeness. Sampling-based motion planners, such as Rapidly-exploring Random Trees (RRT) [2] and Probabilistic Roadmaps (PRM) [3], rely on *probabilistic completeness*, which assures a solution, if one exists, as the number of samples approaches infinity. Sampling-based motion planners sample a set of states from the configuration space (C-space) and check their connectivity without ever explicitly constructing any obstacles. This can reduce computation time considerably, especially as environments increase in complexity. Their performance, however, hinges on the distribution from which points in the C-space are sampled. Uniform samplers can fail in common situations, such as in Figure 1, where the robot needs to traverse narrow regions of measure close to zero under a uniform density in the C-space.

In this work, we propose a new version of sampling-based motion planners with associated learning paradigms that inherit the probabilistic completeness properties of RRTs and PRMs, and are designed to be able to utilize learned sampling distributions. In particular, our Learn and Link (LL) suite of planners can utilize learned information about *critical regions* of the C-space, which are less likely to be sampled under a uniform distribution (e.g., narrow corridors [4]) but are critical for solutions since most solutions for a given, desired class of problems pass through them. This notion relates to the notion of *landmarks*, or parts of the state space that are necessary for reaching the goal in discrete planning problems [5]. However, critical regions are not only useful for reaching the goal, but are also less likely to be reached under a stochastic search paradigm.

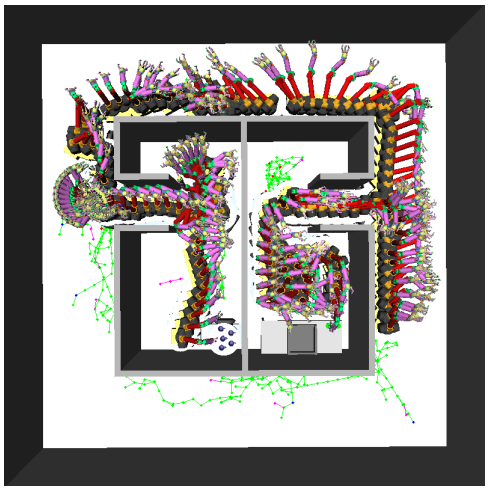

Fig. 1. 7-DOF Barrett WAM arm on a movable base solves a transportation task using LL-RM. The pink points are states that were created when linking the start and goal configurations to the roadmap.

Naive approaches for using critical regions (either learned or hand-coded) in existing sampling-based motion planners tend to fail: merely increasing the probability of sampling from those such regions does not improve the performance of RRT planners because the parts of the tree(s) closest to the critical regions tend to be those that are crashing into the walls adjacent to them. Even when allowing a PRM planner to select configurations from the critical regions as vertices in its roadmap, its simple local planner is unable to connect the vertices in the critical regions to those uniformly sampled unless a considerable amount of time is used in the roadmap building process. The LL planners presented in this paper leverage the positives of these planners while having the necessary modifications to properly utilize critical regions.

We also present a new approach for learning critical regions using convolutional neural networks (CNNs) [6], [7]. We show that when used with our LL planners, this approach can lead to immense speedups in motion planning when image-based training data is available for the planning environment. Since our model only gives base poses, when

dealing with higher dimensional planning, we append each configuration pulled from the critical regions with a random, collision-free configuration for the additional DOF values prior to calling our planners. Although this learning pipeline is limited to image-based representations, our planners use critical regions as inputs and can work with any approach that provides an estimate of the critical regions for a given environment. Our approach is advantageous over pure sampling-based planners and pure learners: it leverages learning from experience to outperform sampling-based planners, but avoids the possibility of missing solutions that limits pure imitation learning, and remains probabilistically complete.

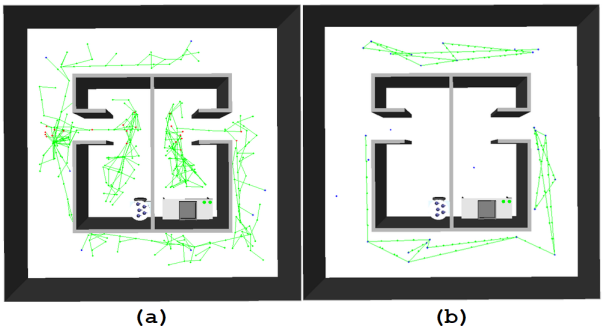

Fig. 2. An example of a roadmap created using LL-RM (a) versus a vanilla PRM (b) for a Barrett WAM arm transportation task. Both planners are given 1 second to build their roadmap using 30 vertices. The green points are states that were created when linking the vertices of the roadmap, the blue points are the vertices of the roadmap that were uniformly sampled, and the red points are the vertices of the roadmap that came from the critical regions. Thus, we can see the extreme difference in C-space coverage using the same amount of time.

The rest of this paper is organized as follows. We begin with a survey of prior work that our approach draws upon in Section II. Section III defines the notion of critical regions; followed by Section IV, which presents the LL planners and discusses their properties. Section V presents our approach for learning critical regions using CNNs. Finally Section VI presents the results of our empirical evaluation on a range of motion planning problems.

II. Related Works

Several methods have been proposed to guiding sampling-based motion planners to solutions. Heuristically-guided RRT [8] uses a probabilistic implementation of heuristic search concepts to create a reasonable bias towards exploration, as well as exploiting known good paths. Although this approach was able to produce less expensive paths, it required a high computational price. Anytime RRTs [9] reuse information from previous RRTs to improve on the path by rejecting samples which have a higher heuristic cost. Batch Informed Trees (BIT*) [10] uses a heuristic to efficiently search a series of increasingly dense implicit random geometric graphs while reusing previous information. In contrast, our method guides our sampling-based motion planners to solutions without the need of a heuristic through leveraging critical regions.

The coupling of learning and MP has been extensively investigated in the past. As discussed in the introduction, naive approaches for using learning to bias sampling in stochastic motion planners don't perform well. Recent work by Ichter et al. uses a Conditional Variational Autoencoder to bias sample points for MP conditioned on encoded environment variables [11]. This encoding is generalizable to higher dimensions, however it requires structuring the data to encompass the state of the robot, the environment, the obstacles (encoded as occupancy grid), and the start and goal configurations. Moreover, during inference, the network model requires this expensive data structuring again, which can take around 50 seconds. In contrast, we focus on image-based learning where data can be easily generated for training using a top-view camera. Moreover, inferences can also be made using a top-view image of the environment in less than 5 seconds. Havoutis et al. use topology to learn sub-manifold approximations that are defined by a set of possible trajectories in the C-space [12]. This requires either motion plans that are generated through a motion capture device, or hand-crafted partial plans. Pan et al. use instance-based learning where prior collision results are stored as an approximate representation of the collision space and the free C-space [13]. This is used to make cheaper probabilistic queries. Although their method shows significant improvement in some environments, their work is limited in finding solutions through narrow passages between obstacles where optimal solution may lie. In our work, the network learns the position of regions that are critical for a given class of MP problems, but have a low probability of getting sampled under a uniform distribution, such as narrow regions. Using the identified critical regions with our LL planners, we show a reduction in average planning time of 57%-99%, and higher success rates, compared to OMPL's RRT, RRT-Connect, and PRM planners.

III. Formal Framework

Given a robot R, an environment E, and a class of MP problems M, we define the *measure of criticality* of a Lebesgue-measurable open set $r \subseteq \mathbb{R}^n$, $\mu(r)$, as $lt_{s_n \to^+ r} \frac{f(r)}{v(s_n)}$, where $f(s_n)$ is the fraction of observed motion plans solving tasks from M that pass through s_n, $v(s_n)$ is the measure of s_n under a reference (usually uniform) density, and $\to^+$ denotes the limit from above along any sequence $\{s_n\}$ of sets containing r ($r \subseteq s_n$ for all n). Note that $\mu(r)$ is zero when $f(r) = 0$. While $\mu(r)$ can be infinite for a region, for all practical purposes we consider regions r with $v(r) > 0$ under the uniform density. Intuitively, regions with high criticality measures are those that are vital for solutions to problems in M, but have a low probability of being sampled under a uniform density.

IV. Learn and Link Planners

In this section we discuss the methods that make up our planners. We describe both the single query planner, LLP, and the multi-query planner, LL-RM. A

Python implementation of the planners can be found at `https://aair-lab.github.io/ll.html`.

A. Learn and Link Planner

LLP is Learn and Link's single query planner. This version differs from LL-RM in that we do not seed our roadmap using vertices that were uniformly sampled (i.e. $m = 0$), and we pass in the start and goal configurations right away to algorithm 1. We do this so that instead of building a general roadmap that spreads across the entire environment, we build a biased roadmap in which subgraphs rooted from the start and goal configurations are connected using additional subgraphs rooted at critical regions to speed up single query planning.

We first describe algorithm 1 in LLP mode. In lines $14-17$, n random collision-free configurations are added as vertices to the roadmap from the critical regions identified by the model. In lines $18-21$, $m = 0$ configurations are added as vertices to the roadmap using a uniform sampler. Since we are in LLP mode, in lines $22-26$, subgraphs rooted from the start and goal configurations are added to the roadmap. For the remainder of the algorithm, we attempt to link the subgraphs spawned from the vertices in the roadmap. In line 28, a random sample is taken to grow the current subgraph in its direction. In line 29, an attempt is made to extend the current subgraph to q_{new}, a new configuration in the direction of q_{rand}. If adding q_{new} to the graph results in a collision, *i.e.* *EXTEND* returns $Trapped$, q_{new} is not added to the graph; otherwise it is added. In line 30, a connectivity attempt occurs to link the current subgraph to the remaining graphs in the roadmap; once all the subgraphs have been connected, *Linked* is returned and the roadmap is complete. By this point, since we are in LLP mode, the start and goal configurations have been linked into a single graph. To extract a path P connecting both points we use Dijkstra's algorithm [14] in line 32. If the conditions in lines $29-30$ are not satisfied, we shift to the next subgraph in the roadmap, using a round-robin approach, in line 36. If an explicit sample cap is reached, *i.e.* $S \neq \infty$, without a solution path being found, an empty path, indicating a failure, is returned.

Algorithm 2 is used in an attempt to link a subgraph to the remaining graphs in the roadmap (lines $9-11$), to remove dead graphs from consideration (line 12), and to check whether all the subgraphs in the roadmap have been linked (line 13). A subgraph is considered dead once it has been linked and added to another graph. Once only one graph remains in the roadmap list, *Linked* is returned to indicate that the roadmap is connected.

Algorithms 3 and 4 are methods reused and adapted from RRT-Connect to work with graphs instead of trees. They are used to grow the current subgraph in the direction of the random samples taken.

B. Learn and Link Roadmap

LL-RM is Learn and Link's multi-query planner. This version differs from LLP in that we attempt to build a general roadmap which can be reused multiple times for traversing a C-space based on collision-free configurations from the critical regions, as well as some uniformly sampled. To solve a query, we simply try to connect the start and goal configurations to the roadmap given by algorithm 1 in LL-RM mode. If we are successful, we use Dijkstra's algorithm to obtain a plan.

When in LL-RM mode, the linking process works the same as LLP. The only differences in LL-RM is that we include additional vertices in the roadmap from areas which were uniformly sampled (i.e. $m > 0$) in lines $18-21$, and we return the roadmap *RM*, instead of a path, when the subgraphs are connected in line 35.

Algorithm 5 is the planning component of LL-RM. In lines $9-12$, two subgraphs are initialized from the start (q_{start}) and goal (q_{goal}) configurations in an attempt to connect them to the existing roadmap *RM*. In lines $13-18$, the same approach used in the building process is employed to connect the start and goal subgraphs to the roadmap. In line 16, a solution check occurs. If a solution is found, the path P connecting the start and goal configurations is obtained using Dijkstra's algorithm in line 17.

C. Probabilistic Completeness

The LL planners maintain the probabilistic completeness property inherent to sampling-based motion planners. Since

Algorithm 1 `Learn and Link`

```
1:  Input
2:      N:     number of critical region states to include
3:      M:     number of uniform states to include
4:      CR:    list of critical region points
5:      Mode:  planner mode; LLP or LL-RM
6:      Q_start: start configuration, if LLP mode
7:      Q_goal:  goal configuration, if LLP mode
8:  Output
9:      P:     collision-free path from q_start to q_goal, if it exists
10:     RM:    constructed roadmap
11: procedure LL(N, M, CR, MODE, Q_start, Q_goal)
12:     curr ← 0
13:     RM ← []
14:     for n = 0 to N − 1 do
15:         s ← SAMPLE(CR)
16:         G_n.init(s)
17:         RM.append(G_n)
18:     for m = 0 to M − 1 do
19:         s ← SAMPLE()
20:         G_{N+m}.init(s)
21:         RM.append(G_{N+m})
22:     if mode == LLP then
23:         G_{N+M}.init(q_start)
24:         G_{N+M+1}.init(q_goal)
25:         RM.append(G_{N+M})
26:         RM.append(G_{N+M+1})
27:     for s = 1 to S do
28:         q_rand ← UNIFORM()
29:         if EXTEND(G_curr, q_rand) ≠ Trapped then
30:             if LINK(RM, G_curr, q_new) == Linked then
31:                 if mode == LLP then
32:                     P ← PATH(RM[0])
33:                     Return P
34:                 else
35:                     Return RM
36:         G_curr ← SWAP(RM, G_curr)
37:     Return []
```

Algorithm 2 LINK

Input
  RM: roadmap of graphs to be connected
  G_{curr}: current subgraph being grown
  Q_{new}: most recent configuration added to G_{curr}
Output
  S: status of G_{curr}'s link attempt
procedure LINK(RM, G_{curr}, Q_{new})
  $R \leftarrow []$
  for G_i **in** $RM \setminus G_{curr}$ **do**
    if $CONNECT(G_i, q_{new}) == Reached$ **then**
      $R.append(G_i)$
  $RM.link\_and\_remove(R, G_{curr})$
  if $|RM| == 1$ **then**
    $S \leftarrow Linked$
  else if $|R| > 0$ **then**
    $S \leftarrow Connected$
  else
    $S \leftarrow Advanced$
  Return S

Algorithm 3 CONNECT

Input
  G: graph being grown towards q
  Q: configuration which G is trying to connect to
Output
  S: status of G's connect attempt
procedure CONNECT(G, Q)
  repeat
    $S \leftarrow EXTEND(G, q)$
  until $S \neq Advanced$
  Return S

LLP and LL-RM only add a finite set of points to seed their roadmaps, the added critical region vertices do not reduce the set of support (regions with non-zero probability) of its uniform sampler, and thus, this property is preserved.

D. Distinction Between LL Planners and PRM

LLP and LL-RM work analogously to PRM, but they have their differences. Essentially, PRM views each initial random configuration as a vertex of a larger graph, whereas LLP and LL-RM view each initial configuration as the start of its own graph. Also, PRM uses a quick and simple local planner to connect its vertices, so they do not store their local plans; whereas LLP and LL-RM use a tree-based local planner inspired by RRT-Connect which saves all the spawned branches when connecting two vertices and incorporates them into the roadmap. Seeing as we store the trees spawned by the local planner, instead of simply attempting to connect a vertex to others within a neighborhood like in PRM, we leverage the additional connectivity and attempt to connect each subgraph to every other non-connected subgraph. These differences are necessary so that subgraphs spawned from the critical regions can be linked to those outside these regions. We discovered that since the critical regions are in areas that are unlikely to be reached under a stochastic search paradigm, it was not enough to use a simple local planner to link the two types of subgraphs; but once they were linked, they drastically sped up the planning process.

Algorithm 4 EXTEND

Input
  G: graph being grown towards q
  Q: configuration which G is stepping toward
Output
  S: status of G's extend attempt
procedure EXTEND(G, Q)
  $S \leftarrow Trapped$
  $q_{near} \leftarrow NN(q, G)$
  if $CONFIG(q, q_{near}, q_{new})$ **then**
    $G.add\_vertex(q_{new})$
    $G.add\_egde(q_{near}, q_{new})$
    if $q_{new} == q$ **then**
      $S \leftarrow Reached$
    else
      $S \leftarrow Advanced$
  Return S

Algorithm 5 LL-RM PLAN

Input
  Q_{start}: start configuration
  Q_{goal}: goal configuration
  RM: roadmap created using LL in LL-RM mode
Output
  P: collision-free path from q_{start} to q_{goal}, if it exists
procedure RM-PLAN(Q_{start}, Q_{goal}, RM)
  $curr \leftarrow 0$
  $G_1.init(q_{start})$
  $G_2.init(q_{goal})$
  $RM.append(G_1)$
  $RM.append(G_2)$
  for $s = 1$ **to** S **do**
    $q_{rand} \leftarrow UNIFORM()$
    if $EXTEND(G_{curr}, q_{rand}) \neq Trapped$ **then**
      if $LINK(RM, G_{curr}, q_{new}) == Linked$ **then**
        $P \leftarrow PATH(RM[0])$
        Return P
    $G_{curr} \leftarrow SWAP(RM, G_{curr})$
  Return []

V. Learning Critical Regions

To learn critical regions, we use an image-based approach. This consists of two phases: a data generation phase and a model training phase.

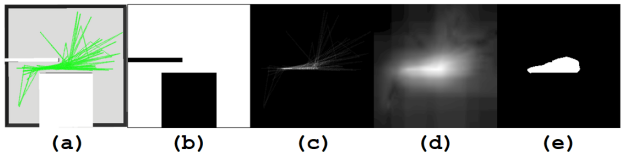

Fig. 3. (a) An example training environment overlain with motion traces. (b) Model input obtained post raster scan. (c) Motion trace image based on μ-criticality of each pixel. (d) Saliency map obtained from the motion trace image. (e) Label obtained after binning the saliency map based on pixel intensity.

A. Data Generation

For each instance of an environment, we begin by randomly selecting a set of 50 motion planning problems from M $\{\Pi_1, ..., \Pi_{50}\}$ and running an off-the-shelf motion planner to generate a corresponding set of motion plans $\{\tau_1, ..., \tau_{50}\}$. We do this multiple times for each handmade environment (see Figure 5) to make sure we fully cover its

critical regions; 179 instances per environment in our dataset. In our data generation process, we utilize an OpenRAVE [15] implementation of OMPL's RRT-Connect planner by `https://github.com/personalrobotics`, though any motion planner can be used instead.

We construct the 224x224 training images for each instance using a raster scan and a saliency model. We describe the process for an *SE(2)* robot (see Figure 3), though it can be extended to mobile manipulators, such as the Barrett arm on a mobile base. We begin by creating a pixel-sized obstacle based on the dimensions of the desired image and the bounds of a given environment. We proceed by scanning the pixel-sized obstacle across the environment. For the input images, if a collision is detected with an environment's obstacles, we select a black pixel, otherwise a white pixel is selected. For the motion trace images, we assign a pixel value based on the μ-criticality of the region the pixel encompasses, which we obtain using $\{\tau_1, ..., \tau_{50}\}$. We then use an implementation of Itti's saliency model [16] by `https://github.com/mayoyamasaki` to extract relevant salient information and smooth out the salient areas from the motion trace images. The saliency maps are binned into two categories, high saliency (denoted by white pixels) and low saliency (denoted by black pixels), and are used as the labels.

Even large environments do not affect training. Since our input images are simple black and white binary images, we are able to convert an environment into a 224x224 image during preprocessing without losing important information. If there exists an environment so large and detailed that too much information is lost when converting it to a 224x224 image, we instead crop the image into smaller components before training/inference, and then stitch the pieces together when looking at the environment as a whole.

The code used to generate the data can be found at `https://aair-lab.github.io/ll.html`.

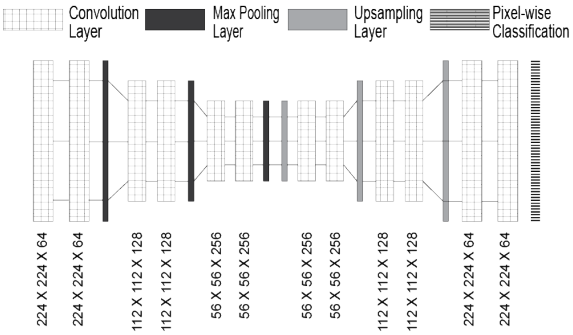

Fig. 4. Network architecture selected for our model.

B. Network Architecture

We propose a general structure for a convolutional encoder-decoder neural network which learns to detect critical regions.

Our network, depicted in Figure 4, has 14 convolutional layers. 7 layers in the encoder network and 7 layers in the decoder network forming the encoder-decoder architecture for pixel-wise classification. A max pooling layer with stride 2 is introduced after each group of same number of filters to encode the learned representation. Similarly, an upsampling layer is added before each deconvolutional layer group of same number of filters. We draw inspiration from [6] for a learnable upsampling layer in the decoder network.

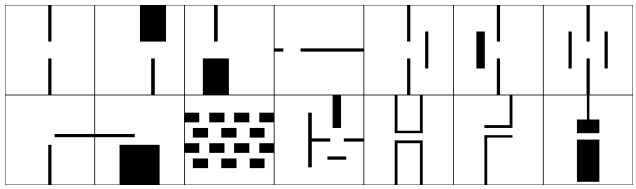

Fig. 5. Handmade training environments used to generate data (not including rotations).

The first two convolutional layers have 64 filters with a 3×3 kernel. Motivated by recent promising results [17], we stack 3 layers with 3×3 kernel size to obtain a similar receptive field as a 7×7 kernel, with 81% less parameters, and more effective training owing to the added non-linearity after every layer. For the initial layer group of filter size 64 and 128, we stack only two layers of kernel size 3×3. Though the receptive field is smaller than a 7×7 kernel, we still stack only 2 layers as our problem statement does not require learning complex geometric features. The next 2 layers are of 128 filters with a 3×3 kernel. We add 3 layers of 256 filters each, with a 3×3 kernel, for a larger receptive field since deeper layers learn invariant complex features [18]. All the convolutional and max-pool layers have padding added to them.

In the decoder network, corresponding deconvolutional layers to the encoder network are used. The upsampled output is used for pixel-wise classification using a softmax cross-entropy loss function. Each layer in the network is activated using ReLu nonlinearity.

C. Training

The network was trained using a mini-batch size of 16 and a dataset of 10,024 images. Following [19], we did not train the network with dropout [20] since the output of every layer is batch-normalised, which also acts as a regularizer. We use Adam Optimizer [21] with a 0.1 learning rate to train the network. The network was trained for 50,000 epochs since the loss converges at this point. The training images are shuffled before each epoch and trained with mini-batch to ensure that every input to the network is different from the previous. This assists the optimizer to exit local minima. We used an implementation of SegNet [6] by `https://github.com/andreaazzini` for its data pipelines since they provide a fast and efficient input pipeline which reduces training time.

On average, training for the full dataset takes approximately 3 hours on a single Nvidia GTX 1080Ti.

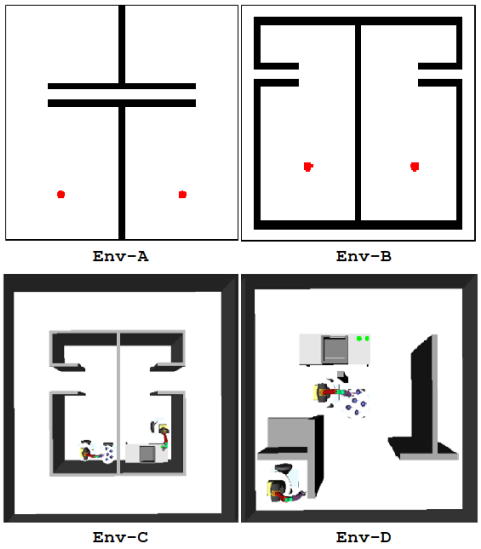

Fig. 6. Env-A and Env-B are the *SE(2)* used to evaluate the model. Red dots represent the start and goal configurations. Env-C and Env-D are the 10-DOF test environments used to evaluate the model. The robot is placed at the start and goal configurations. Test environments are unseen to the network (they were not used in the training set of problems).

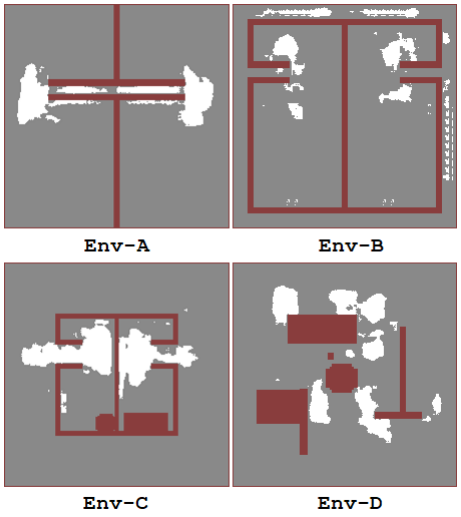

Fig. 7. Network output for the test environments shown in Figure 5. The μ-criticality of the output is as follows: μ(Env-A) = 0.604, μ(Env-B) = 1.351, μ(Env-C) = 0.538, and μ(Env-D) = 0.398.

VI. Empirical Evaluation

In this paper, we focus on investigating two main questions:

1) Can CNNs be used to identify critical regions for motion planning?
2) Can critical regions be used to improve planning performance?

The first consideration aims to see if we can extend the visual prowess exhibited by CNNs to identifying the critical regions of an environment. The second consideration aims to see if knowing critical regions helps a planner reduce its computation time. Our intent is not to create the best, most optimal planner, but to evaluate the gains that can be made when a planner properly leverages the critical regions of the C-space being traversed.

To investigate these considerations, we designed challenging MP problems for *SE(2)* and the Barrett WAM arm (see Figure 6), and we explored various network architectures. We evaluate the quality of the critical regions given by our CNN using their measure of criticality μ (see Figure 7), as well as the planning time used by our planners. For our planning problems, 100 MP problems were constructed using the same start and goal pair, the same range, and a planning time limit of 60 seconds. LLP and LL-RM both use 5% of the critical regions identified as n, and m is 0 and $n/10$, respectively. OMPL PRM and LL-RM are both given 1 second to build a roadmap prior to planning. Our approach is for robots with omnidirectional base movements, though any movement constraints can be added in the *EXTEND* module. It is important to note that OMPL is written in highly optimized C++ code compared to our Python implementation.

A. Evaluating Identified Critical Regions

We evaluate the critical regions identified by a model for an environment using its ground truth motion trace image (see Figure 3(c)). We first cluster the model-identified critical regions using k-Nearest Neighbors [22] with $k = 25$. Then we evaluate each critical region cluster c_i using its μ-criticality, where we estimate $v(c_i)$ as the area of the pixels in the cluster. The metric values for each cluster are then summed to obtain an evaluation of the environment as a whole. The higher the value, the better the critical regions.

We use this metric instead of comparing pixel accuracy with the ground truth label since the motion trace image is embedded with much more information regarding the quality of the critical regions than simply identifying them.

Figure 9 shows a comparison of the critical regions identified by VGGNet, SegNet, and our parsimonious network using this metric.

B. Results

Our results suggest that both LLP and LL-RM require far less time to obtain a solution than OMPL's RRT, RRT-Connect, and PRM planners, especially as the environments increase in difficulty. Figure 8 shows a comparison of planning time used by the OMPL planners and our LL planners using the areas learned by our parsimonious network.

1) SE(2) Domain: For *SE(2)*, LLP and LL-RM outperformed OMPL's planners in terms of average planning time and success rate. On Env-A, RRT, RRT-Connect, and PRM had success rates of 19%, 0%, and 53%, respectively. For successful plans, LLP and LL-RM required 97% and 99% less time on average, respectively, than PRM, the best performing OMPL planner on this environment. On Env-B, RRT, RRT-Connect, and PRM had success rates of 93%, 8%, and 100%, respectively. For successful plans, LLP and LL-RM required 64% and 74% less time on average, respectively, than PRM, the best performing OMPL planner on this environment.

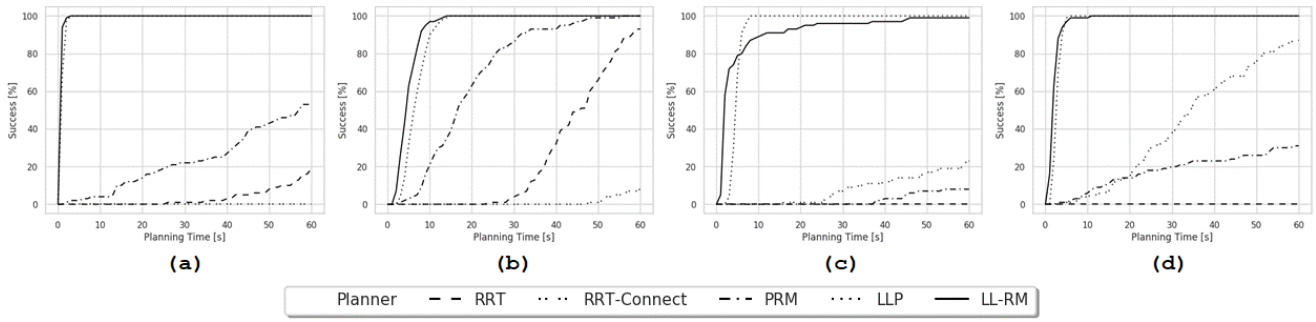

Fig. 8. (a) and (b) show the percentage of trials solved versus planning time for the 100 different trials for the *SE(2)* domain, and (c) and (d) for the 10-DOF domain. Environments A-D are shown in Figure 6.

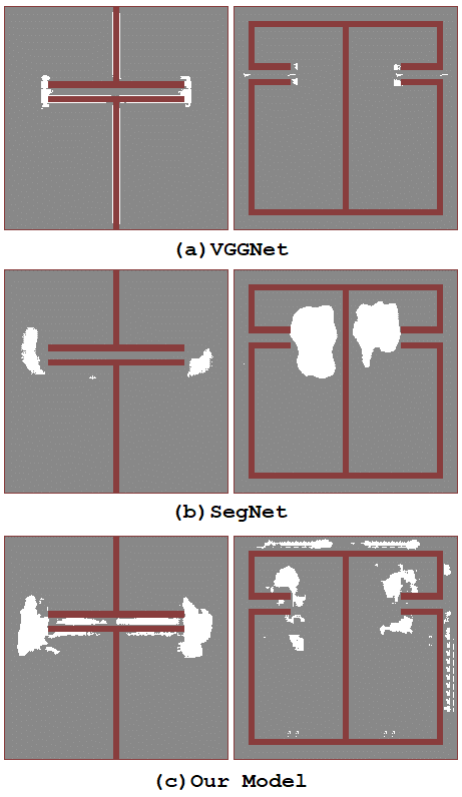

Fig. 9. (a) Critical regions identified using VGGNet. From left to right, their μ-criticality is 0 and 0. (b) Critical regions identified using SegNet. From left to right, their μ-criticality is 0 and 0.260. (c) Critical regions identified using our network. From left to right, their μ-criticality is 0.604 and 1.351.

2) 10-DOF Domain: For the transportation tasks using the movable Barrett arm, LLP and LL-RM require less planning time on average and had higher success rate than OMPL. On Env-C, RRT, RRT-Connect, and PRM had success rates of 0%, 87%, and 31%, respectively. When comparing successful plans, LLP and LL-RM required 89% and 92% less time on average, respectively, than PRM, the best performing OMPL planner on this environment. On Env-D, RRT, RRT-Connect, and PRM had success rates of 0%, 23%, and 8%, respectively. When comparing successful plans, both LLP and LL-RM required 88% less time on average than RRT-Connect, the best performing OMPL planner on this environment.

C. Network Ablation Study

Since obstacles in an environment can be represented by bounding boxes, most of the objects in our dataset have regular geometric shapes. We performed an ablation study to find the simplest model that can learn the feature representation using as few layers as possible, without compromising the results. We investigated two different types of neural networks and compared their performance with SegNet using our μ-criticality measure. The ablation study for both types of architecture is discussed below.

1) Convolutional Network: The main question in the network ablation study was to enquire whether a solely convolutional network would suffice in solving this problem.

The CNN-based VGGNet learned only to trace obstacle borders. The μ-criticality for VGGNet as shown in the Figure 9(a) is 0 for all the test environment. Although the criticality values were not promising, it still shed light on network behaviour. The network was able to learn the geometry of the obstacles in the image, which CNNs are known to be good at, but was unable to identify the critical regions. Moreover, training VGGNet takes 16 hours on a single Nvidia GTX 1080Ti GPU for 50,000 epochs.

2) Encoder-Decoder: After a fully convolutional approach failed, we investigated how well a segmentation architecture, such as SegNet, could learn critical regions. Following promising initial results using SegNet as shown in Figure 9(b), we investigated an encoder-decoder network which can learn the latent representation in a supervised manner for pixel-wise classification. In an encoder-decoder, the encoder can learn the feature representation and encode it into a latent space, while the decoder can learn the pixel-wise classification on the learned features.

A simple encoder-decoder network with 4 layers each in the encoder and decoder sections of the network was able to somewhat learn the critical regions of the data well, obtaining μ-criticality scores of 0.0156 and 1.043, respectively on the *SE(2)* environments, but tended to show a lot of checkerboard artifacts in the identified regions.

Building on the simple encoder-decoder architecture, we added 3 more batch normalized layers to increase the recep-

tive field size in an attempt to smooth out the critical regions and generalize to the test set. We achieved μ-criticality scores of 0.604 and 1.351 for respective environments as shown in Figure 9(c), indicating the network's ability to identify the critical regions for motion planning.

VII. Conclusions

We presented a new approach in learning for MP and used it to create a new suite of sample-based motion planners, Learn and Link. We constructed a fully convolutional encoder-decoder neural network to learn critical regions for MP problems that generalizes across different domains. Our model is used by our LL planners to remedy the limitations of uniform sampling, without compromising guarantees of correctness.

Our results on challenging MP problems demonstrate that CNNs have the capability to extract important features relevant to MP problem.

ACKNOWLEDGMENT

This work was supported in part by the NSF under grants IIS 1844325 and IIS 1909370.

References

[1] J. H. Reif, "Complexity of the mover's problem and generalizations," in *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*. IEEE, 1979, pp. 421–427.

[2] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, 2001.

[3] P. Svestka, J. Latombe, and L. Overmars Kavraki, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[4] S. R. Lindemann and S. M. LaValle, "Current issues in sampling-based motion planning," in *Robotics Research. The Eleventh International Symposium*. Springer, 2005, pp. 36–54.

[5] J. Hoffmann, J. Porteous, and L. Sebastia, "Ordered landmarks in planning," *Journal of Artificial Intelligence Research*, vol. 22, pp. 215–278, 2004.

[6] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *arXiv preprint arXiv:1511.00561*, 2015.

[7] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[8] C. Urmson and R. Simmons, "Approaches for heuristically biasing rrt growth," in *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 2. IEEE, 2003, pp. 1178–1183.

[9] D. Ferguson and A. Stentz, "Anytime rrts," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE, 2006, pp. 5369–5375.

[10] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 3067–3074.

[11] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7087–7094.

[12] I. Havoutis and S. Ramamoorthy, "Motion synthesis through randomized exploration on submanifolds of configuration space," in *Robot Soccer World Cup*. Springer, 2009, pp. 92–103.

[13] J. Pan, S. Chitta, and D. Manocha, "Faster sample-based motion planning using instance-based learning," in *Algorithmic Foundations of Robotics X*. Springer, 2013, pp. 381–396.

[14] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[15] R. Diankov and J. Kuffner, "Openrave: A planning architecture for autonomous robotics," *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*, vol. 79, 2008.

[16] L. Itti and C. Koch, "A saliency-based search mechanism for overt and covert shifts of visual attention," *Vision research*, vol. 40, no. 10-12, pp. 1489–1506, 2000.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[18] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[19] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[20] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[22] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.