Learning Critical Regions for Robot Planning Using Convolutional Neural Networks

Daniel Molina and **Kislay Kumar** and **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 (CNN) can be used to identify critical regions for motion planning problems.

We also introduce a new sampling-based motion planner, *Learn and Link*. Our planner leverages 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.

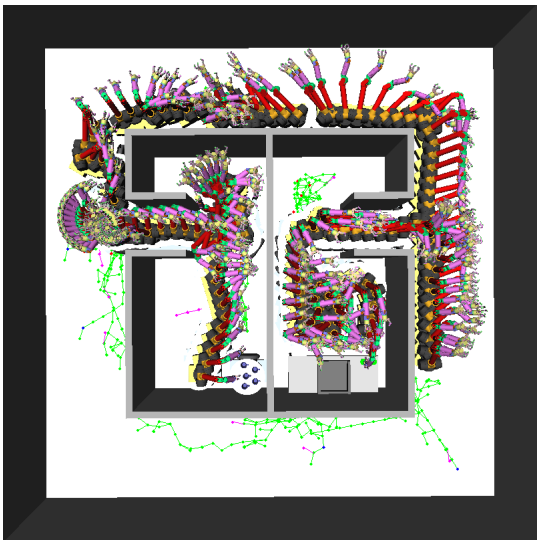


Figure 1: 7-DOF Barrett WAM arm on a movable base solves a transportation task using LL-RM. The pink points are states that were created when linking the start and goal configurations to the roadmap.

Introduction

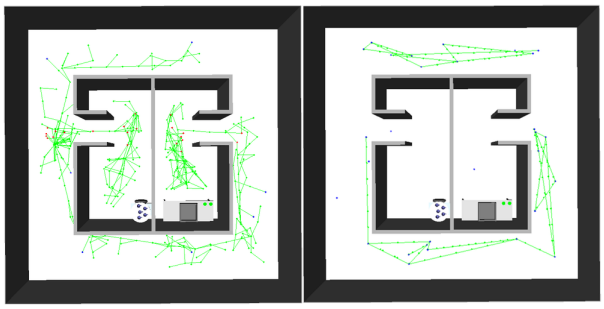
The MP problem deals with finding a feasible trajectory that takes a robot from a start configuration to a goal configuration without colliding with obstacles. From a computational complexity point of view, even a simple form of the MP problem is NP-hard (Reif 1979). In order to achieve computational efficiency, motion planning methods relax requirements of completeness. Sampling-based motion planners, such as Rapidly-exploring Random Trees (RRT) (LaValle and Kuffner Jr 2001) and Probabilistic Roadmaps (PRM) (Svestka, Latombe, and Overmars Kavraki 1996), 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 two main considerations: the way the C-space is sampled, and the particular order in which samples are chosen.

In order to improve the scalability of MP we present a new approach for learning approximate, significant landmarks, or critical regions, for MP problems. These regions are those that are less likely to be sampled, such as narrow corridors (Lindemann and LaValle 2005), but are critical for solutions since most solutions to a given class of problems pass through them. The notion of discrete landmarks has been used to improve the performance of classical planners (Hoffmann, Porteous, and Sebastia 2004). Our approach relates to this concept but differs in its consideration of the sets of states that are not only useful for reaching the goal, but are also less likely to be reached under a stochastic search paradigm. Further, unlike landmarks, critical regions are not necessary parts of a solution.

In this work, we overcome the sampling limitations of sampling-based motion planners using a CNN to identify critical regions prior to planning. Recent work on CNNs has demonstrated their utility in situations where the input data can be expressed as an image-based representation (Badrinarayanan, Kendall, and Cipolla 2015; Ronneberger, Fischer, and Brox 2015). For a MP problem, we can create an image representation through recording the motion plans and environment. Our approach begins using RRT-Connect

(Kuffner and LaValle 2000) to compute motion plans on a set of handmade training environments, though historical data or human demonstrations can be used as well. We proceed using a raster scan to construct images of the environments and trajectories. The environment images are based on collisions, and the trajectory images are based on path intersections. Saliency maps are created from the trajectory images using Itti's saliency model (Itti and Koch 2000), and are then thresholded to identify the most salient regions. Finally, we train the CNN to identify critical regions using the thresholded saliency labels. The model's generalizability is evaluated on two domains: *SE(2)* and a 10-DOF C-space involving a 7-DOF Barrett WAM arm on a moveable platform (see Figure 1). We demonstrate that the learned model successfully generalizes to unseen problem instances.

We also show that our model can be utilized to construct a new sampling-based motion planner, Learn and Link, to reduce the computation necessary to solve challenging MP problems without compromising guarantees of correctness. Learn and Link has two modes: a single query mode, referred to as Learn and Link planner (LLP), and a multi-query mode, referred to as Learn and Link Roadmap (LL-RM). LL-RM is similar to PRM in that we construct a general roadmap, though using both critical regions and random configurations, which covers an environment and can be used for multiple queries (see Figure 2). LLP is used for more efficient single query plans where we do not need a general roadmap, and instead want to bias the construction of a roadmap for a given start-goal pair and set of critical regions. Currently, Learn and Link is best suited for MP problems involving base movements and is being extended for stationary planning problems in the future. We compare our new planner with sampling-based motion planners from OMPL.



(a) (b)

Figure 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.

Our experiments reveal that areas of an environment that are critical for MP, but have a low probability of being sampled under a uniform distribution, are identifiable using CNNs. We demonstrate that these critical regions can be utilized by Learn and Link to robustly compute motion plans while requiring far less planning time than existing sampling-based motion planners. 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.

We believe these results are general, and that they hold across many domains. This approach is particularly useful in situations where one would have prior knowledge on the types of environments being traversed, but not have the luxury of using time-expensive planners. We demonstrate that in situations where some prior knowledge on the distribution of environments and class of planning problems is known, the critical regions identified by our model can reduce planning time considerably. Even considering preprocessing, these costs become increasingly negligible as the dimensionality of the planning problem increases. As demonstrated in the experimental section for the 10-DOF problem, the classical motion planners tend to fail a majority of the time when quick planning is required.

Our approach presents a first step towards creating hierarchies for continuous planning problems by extracting critical regions for a given environment and defining actions as transitions between them.

Related Works

Several methods have been proposed to guiding sampling-based motion planners to solutions. Heuristically-guided RRT (Urmson and Simmons 2003) 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 (Ferguson and Stentz 2006) reuse information from previous RRTs to improve on the path by rejecting samples which have a higher heuristic cost. Batch Informed Trees (BIT*) (Gammell, Srinivasa, and Barfoot 2015) uses a heuristic to efficiently search a series of increasingly dense implicit random geometric graphs while reusing previous information. In contrast, our method guides sampling-based motion planners to solutions without the need of a heuristic. Rather, the learned sampling distribution helps bias sampling towards critical regions which have a lower probability of getting sampled, but in most scenarios are necessary for solutions.

The coupling of learning and MP has been extensively investigated in the past. Recent work by Ichter et al. uses a Conditional Variational Autoencoder to bias sample points for MP conditioned on encoded environment variables (Ichter, Harrison, and Pavone 2018). 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 struc-

turing 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. This results in faster inference for situations demanding faster motion plans. Havoutis et al. use topology to learn sub-manifold approximations that are defined by a set of possible trajectories in the C-space (Havoutis and Ramamoorthy 2009). 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 (Pan, Chitta, and Manocha 2013). 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. These critical regions can be leveraged by any motion planner for faster solutions.

Our work shows a reduction in average planning time of 57%-99%, and higher success rates for quick planning, compared to OMPL's RRT, RRT-Connect, and PRM planners.

Learning Critical Regions

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 exploration under a uniform density.

To learn critical regions, we construct a set D_{train} of N_{train} MP problem instances $\{\Pi_1, ..., \Pi_{N_{train}}\}$ and obtain a corresponding set of solution trajectories $\{\tau_1, ..., \tau_{N_{train}}\}$ to construct the images. A set D_{test} of N_{test} MP problem instances is used to evaluate the learned model. The problem instances are picked for various environments. Raster scans of the environments create the input images, and the solution trajectories are used to construct the label images.

Our approach consists of two phases: a data generation phase and a model training phase.

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 3) to make sure we fully cover its critical areas; 179 instances per environment in our dataset. In our data generation process, we utilize an OpenRAVE (Diankov and Kuffner 2008) 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 4), 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 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.

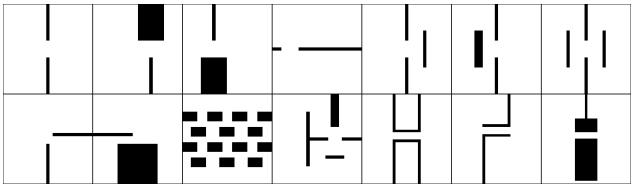


Figure 3: Handmade training environments used for the *SE(2)* domain (not including rotations). Training environments for the Barrett arm are similar, though scaled appropriately for the difference in robot size.

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 5, has 14 convolutional layers. 7 layers in the encoder network and 7 layers in the decoder network forming the encoder-decoder architecture for

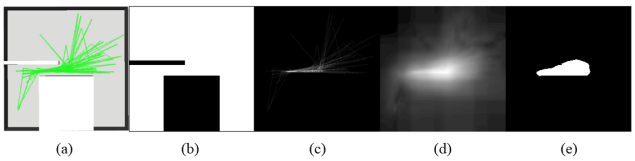


Figure 4: (a) An example training environment overlain with motion traces. Training data for each instance is created using this information. (b) Model input obtained post raster scan. (c) Motion trace image based on μ-criticality of each pixel. (d) Saliency map obtained from the motion trace image. (e) Label obtained after binning the saliency map based on pixel intensity.

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 (Badrinarayanan, Kendall, and Cipolla 2015) for a learnable upsampling layer in the decoder network.

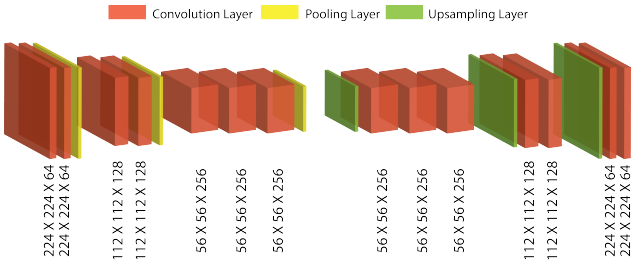


Figure 5: Network architecture selected for our model.

The first two convolutional layers have 64 filters with a 3×3 kernel. Motivated by recent promising results (Simonyan and Zisserman 2014), 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 (Zeiler and Fergus 2014). 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.

Training

The network was trained using a mini-batch size of 16 and a dataset of 10,024 images. Following (Ioffe and Szegedy 2015), we did not train the network with dropout (Srivastava et al. 2014) since the output of every layer is batch-normalised, which also acts as a regularizer. We use Adam Optimizer (Kingma and Ba 2014) 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 (Badrinarayanan, Kendall, and Cipolla 2015) 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.

Processing Critical Regions

In the following section we discuss how to process the model output so that it can be used by Learn and Link.

Seeing as the model output is in image format, we need a mapping of pixel indices to the environment's coordinate system: $f : (i, j) \mapsto [p_{min_x}, p_{min_y}, p_{max_x}, p_{max_y}]$. We define such a mapping as follows:

$$f(i,j) = \begin{vmatrix} p_{min_x} & p_{min_y} \\ p_{max_x} & p_{max_y} \end{vmatrix}$$

$$f(i,j) = \begin{vmatrix} b_{min} & b_{max} \\ b_{min} & b_{max} \end{vmatrix} + \begin{vmatrix} i & -(j+1) \\ i+1 & -j \end{vmatrix} \times p_w$$

where $p_w = \frac{b_{max} - b_{min}}{224}$ and $0 \leq i, j \leq 224$.

In the equation: i is the horizontal pixel index, j is the vertical pixel index, b_{min} and b_{max} are the bounds of our square environment, p_w is the width of a pixel in terms of the environment's coordinate system, and $f(i, j)$ gives the bounds for a pixel located at (i, j) in terms of the environment's coordinate system. i and j are bounded since 224×224 is the desired dimension of the model input. It can be altered to accommodate the model.

Using f, we iterate through the pixels of the model output and store the sample bounds of the pixels identified as critical regions, i.e. the white pixels. We then take a list of critical region points and pass them to Learn and Link.

Learn and Link

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

Learn and Link Planner

LLP is Learn and Link's single query mode. This version differs from LL-RM in that we set $m = 0$ and 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 the process.

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 (Dijkstra 1959) 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 depict methods reused and adapted from RRT-Connect. These methods are used to grow the current subgraph in the direction of the random samples taken.

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 []
```

Learn and Link Roadmap

LL-RM is Learn and Link's multi-query mode. 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 to obtain a plan.

When in LL-RM mode, the linking process works the same as described for LLP. The only differences in LL-RM mode are 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.

Probabilistic Completeness

Link and Learn maintains the probabilistic completeness property inherent to sampling-based motion planners. Since LLP and LL-RM only add a finite set of points to seed their roadmaps, it does not reduce the set of support (regions with non-zero probability) of its uniform sampler, and thus, this property is preserved. Even when the network paired with Learn and Link fails to identify any critical regions, no issue arises. In this scenario, LLP works analogously to RRT-Connect, and LL-RM works analogously to PRM, both of which are probabilistically complete.

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

Extension for Mobile Manipulators

The extension for higher DOF robots follows simply. Since our model only gives base poses, we append each configuration in *CR* with a random, collision-free configuration for the additional DOF values prior to calling the planner. The algorithm then proceeds as usual.

Experiments

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 leverages the critical regions of the C-space being traversed.

To investigate these considerations, we designed challenging MP problems for *SE(2)* (see Figure 6) and the Barrett WAM arm (see Figure 7), and we explored various network architectures. For both domains, 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 an arbitrary local planner can be substituted in the *EXTEND* module. It is also important to point out that OMPL is written in highly optimized C++ code compared to our Python implementation.

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 []

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 4(c)). We first cluster the model-identified critical regions using k-Nearest Neighbors (Altman 1992) 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

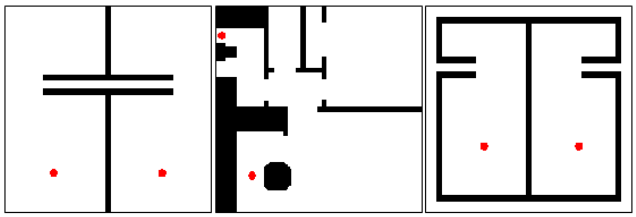


Figure 6: *SE(2)* test environments used to evaluate the model. Red dots represent the start and goal configurations.

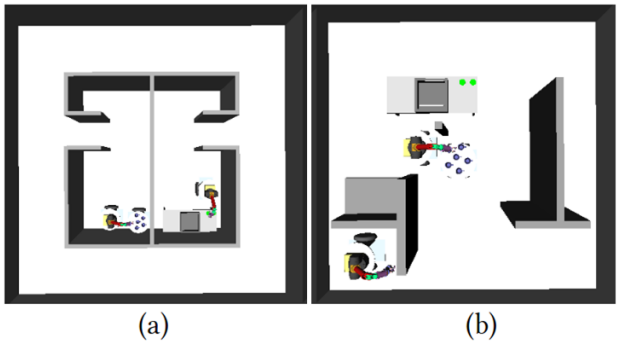


Figure 7: Barrett arm test environments used to evaluate the model. The robot is placed at the start and goal configurations.

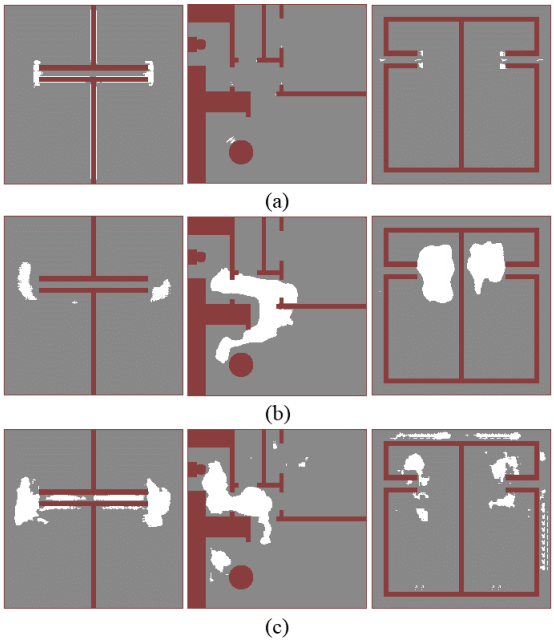


Figure 8: (a) Critical regions identified using VGGNet. From left to right, their μ-criticality is 0, 0, 0. (b) Critical regions identified using SegNet. From left to right, their μ-criticality is 0, 0.141, and 0.260. (c) Critical regions identified using our network. From left to right, their μ-criticality is 0.604, 0.371, and 0.702.

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

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. Figures 9 and 10 show a comparison of planning time used by the OMPL planners and Learn and Link using the areas learned by our parsimonious network.

SE(2) Domain For *SE(2)*, LLP and LL-RM outperformed OMPL's planners in terms of average planning time and success rate. It was only close on environment (b) where LLP and LL-RM required 66% and 57% less time on average, respectively, than PRM, the best performing OMPL planner on this environment; which we attribute to there being a lot more open space and less narrow passages compared to the size of the robot. Also recall that PRM and LL-RM both received an additional second for roadmap construction. On environments (a) and (c), whose passages allow for limited movement, the difference is more extreme. On environment (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 environment (c), 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.

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 environment (a), 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 environment (b), 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.

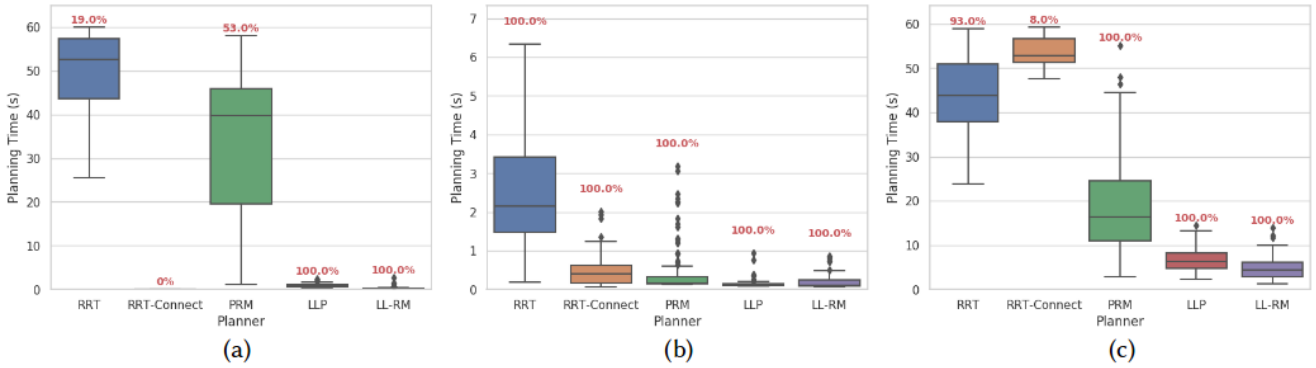


Figure 9: Boxplots of the 100 runs comparing planning time for the *SE(2)* domain. The success rate of the 100 plans for each planner is listed in red on the plot.

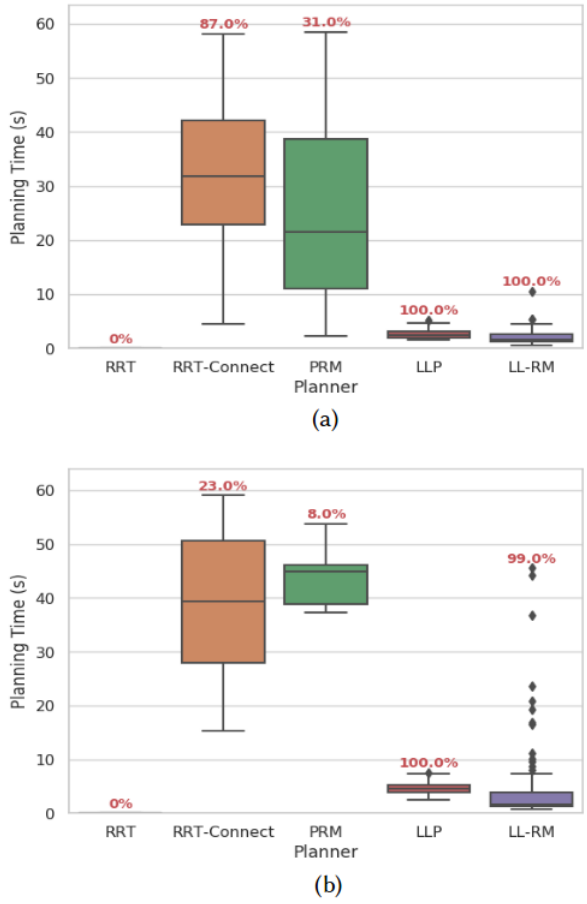


Figure 10: Boxplots of the 100 runs comparing planning time for the 10-DOF domain. The success rate of the 100 plans for each planner is listed in red on the plot.

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.

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 8(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.

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 8(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, 0.384, 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 a μ-criticality score of 0.604, 0.371 and 0.702 for respective environments as shown in Figure 8(c), indicating the network's ability to identify the critical regions for motion planning.

Conclusions

We presented a new approach in learning for MP and used it to create a new sample-based motion planner, 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 Learn and Link 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.

Acknowledgements

This work was supported in part by the NSF under grant IIS 1844325.

References

[Altman 1992] Altman, N. S. 1992. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician* 46(3):175–185.

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

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

[Dijkstra 1959] Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1(1):269–271.

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

[Gammell, Srinivasa, and Barfoot 2015] Gammell, J. D.; Srinivasa, S. S.; and Barfoot, T. D. 2015. 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*, 3067–3074. IEEE.

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

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

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

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

[Itti and Koch 2000] Itti, L., and Koch, C. 2000. A saliency-based search mechanism for overt and covert shifts of visual attention. *Vision research* 40(10-12):1489–1506.

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

[Kuffner and LaValle 2000] Kuffner, J. J., and LaValle, S. M. 2000. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, 995–1001. IEEE.

[LaValle and Kuffner Jr 2001] LaValle, S. M., and Kuffner Jr, J. J. 2001. Randomized kinodynamic planning. *The international journal of robotics research* 20(5):378–400.

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

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

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

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

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

[Srivastava et al. 2014] Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15(1):1929–1958.

[Svestka, Latombe, and Overmars Kavraki 1996] Svestka, P.; Latombe, J.; and Overmars Kavraki, L. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12(4):566–580.

[Urmson and Simmons 2003] Urmson, C., and Simmons, R. 2003. Approaches for heuristically biasing rrt growth. In *Intelligent Robots and Systems, 2003.(IROS 2003). Pro-*

ceedings. 2003 IEEE/RSJ International Conference on, volume 2, 1178–1183. IEEE.

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