Neural Collision Clearance Estimator for Fast Robot Motion Planning

J. Chase Kew\textsuperscript{1}  
Brian Ichter\textsuperscript{1}  
Maryam Bandari\textsuperscript{2}  
Tsang-Wei Edward Lee\textsuperscript{1}  
Aleksandra Faust\textsuperscript{1}

Abstract—Collision checking is a well known bottleneck in sampling-based motion planning due to its computational expense and the large number of checks required. To alleviate this bottleneck, we present a fast neural network collision checking heuristic, ClearanceNet, and incorporate it within a planning algorithm, ClearanceNet-RRT (CN-RRT). ClearanceNet takes as input a robot pose and the location of all obstacles in the workspace and learns to predict the clearance, i.e., distance to nearest obstacle. CN-RRT then efficiently computes a motion plan by leveraging three key features of ClearanceNet. First, as neural network inference is massively parallel, CN-RRT explores the space via a parallel RRT, which expands nodes in parallel, allowing for thousands of collision checks at once. Second, CN-RRT adaptively relaxes its clearance threshold for more difficult problems. Third, to repair errors, CN-RRT shifts states towards higher clearance through a gradient-based approach that uses the analytic gradient of ClearanceNet. Once a path is found, any errors are repaired via RRT over the misclassified sections, thus maintaining the theoretical guarantees of sampling-based motion planning. We evaluate the collision checking speed, planning speed, and motion plan efficiency in configuration spaces with up to 30 degrees of freedom. The collision checking achieves speedups of more than two orders of magnitude over traditional collision detection methods. Sampling-based planning over multiple robotic arms in new environment configurations achieves speedups of up to 51\% over a baseline, with paths up to 25\% more efficient. Experiments on a physical Fetch robot reaching into shelves in a cluttered environment confirm the feasibility of this method on real robots.

I. INTRODUCTION

Motion planning is the problem of finding a continuous, collision-free path between a start state and a goal state. It is often considered in the configuration space (C-Space), the set of possible robot poses [24]. As the dimensionality of this space increases, solving the planning problem exactly quickly becomes intractable. Sampling-based algorithms address the issue by forming an approximate, implicit representation of the C-Space through a set of probing samples, connected locally by querying a collision checker. The primary algorithmic primitives are thus: (1) sampling the free state space, (2) identifying nearest neighbors for each sample, (3) local steering to connect samples, and (4) collision checking to verify the local connection is free of obstacles. The collision check is generally considered the primary bottleneck in sampling-based motion planning, consuming up to 90\% of the total computation time [11].

Classical approaches to collision checking rely on computing the geometry of all objects in the scene. Though there exists a rich literature on accelerating collision checking through parallelization, broad and narrow search phases and other optimizations, the exact geometric collision check remains a computationally expensive operation. Researchers have attempted to reduce the number of required collision checks [4] by certifying regions to be collision free or by using probabilistic collision checks [23] to provide a quick, approximate belief about which regions of the C-Space are free and which are in collision.

This work approaches the bottleneck of collision checking and accelerates sampling-based planning with three key insights. First, collision status in a known environment can be determined given the position of all mobile bodies. Second, both sampling-based motion planning and neural network inference are embarrassingly parallelizable. Third, it is easier to verify and repair a partially-correct trajectory than to plan from scratch.

The first insight leads to the idea of learning to predict collision from examples, without the robot explicitly knowing either its own geometry or the obstacles’\textsuperscript{1}. With many examples, even without sensing, the robot can learn to predict the clearance to the nearest object. This is because nearby poses are likely to have similar clearances. Deep neural networks are universal approximators that with enough data can learn any continuous function [16], such as the minimum clearance. We train a deep neural network, ClearanceNet, to predict collision clearance for robots and environments with many degrees-of-freedom (DoF), using geometry for training but not for inference.

The second insight, that sampling-based methods are embarrassingly parallelizable, is noted in [2], [3]. Leveraging neural networks’ efficiency at processing large batches, we collision check entire edges together and expand from many
nodes in parallel. This batch processing leads to a heuristic collision check two orders of magnitude faster than optimized geometric methods.

The third insight is relevant because, although very fast, the learned collision heuristic is approximate and can make mistakes. For this reason, we build on [9] to rapidly verify and repair motion plans with a two step process. First, we use the gradient of the neural network to shift poses towards higher clearance. Second, we repair remaining misclassified poses with traditional planning.

Building on these insights, this paper makes three contributions to achieve fast motion planning for high-DoF robots. The first, ClearanceNet, is a neural network that estimates minimum clearance between bodies, conditioned on the state of each movable body in the workspace. ClearanceNet is trained using large-scale hyperparameter tuning that simultaneously searches for the appropriate training hyperparameters and trains the clearance estimator. The second contribution is these hyperparameters, which were found to be consistent across environments and robots. The third contribution is CN-RRT, an adaptive and massively parallel extension of the Fastron-RRT algorithm [9] that effectively leverages massive parallelism of neural network inference (batching up to 60 edges together and 5,000 collision checks). CN-RRT includes a flexible repair step that first uses a gradient-based path update with the ClearanceNet gradient to move the path away from obstacles, followed by a traditional and theoretically-backed repair over the remaining misclassified regions. ClearanceNet and CN-RRT are evaluated on five different environments (Figure 2) with obstacles including movable shelves and convex and concave geometric surfaces. The environments range from 7 to 30 robot degrees of freedom, with robots including multiple arms with fixed bases, mobile manipulators, and a full bodied Fetch robot. We compare the presented method with two baselines: a traditional geometric collision checker (Gilbert-Johnson-Keerthi or GJK [13]) implemented in PyBullet [7], and FASTRON [9]. ClearanceNet performs collision checking up to 845x faster in sufficiently large batches, and reaches accuracy of 91-96%. CN-RRT both completes planning faster (up to 51%) and produces up to 25% cheaper paths.

II. RELATED WORK

Several previous works have used tools from machine learning to offset the cost of collision checking. One approach seeks to identify and evaluate only important regions of the robot’s state space, thus reducing the number of required collision checks. Some methods learn heuristics for promising paths [25], [5], learn a distribution of promising regions [18], or evaluate only the most promising edges [6]. Our approach, which learns to accelerate the actual collision checking procedure, is complementary and can be used in conjunction with these approaches.

Other lines of work use learning to accelerate collision checking itself. This has been done by approximating C-Space with Gaussian mixture models [17]. Other works learn to classify collisions based on the results of nearby collision checks, leveraging the fact that nearby configurations tend to share the same collision state [23], [28], or train a support vector machine offline and actively adapt it online to predict the penetration depth of rigid bodies [29]. Several neural network-based approaches estimate collision probability for a moving robot: for drone flying [26], based on the uncertainty of the environment, [20], or using joint feedback to detect collisions in real-time [15]. Although based on neural networks, these methods use sensor data or force feedback to estimate the collision during motion. In contrast, our method does not require any sensor information and considers purely geometry based collisions. [19] learns to predict collision from an input environment and latent state. In contrast, our method focuses specifically on learning to classify collisions in the full C-Space and ultimately certifies the computed motion plan is collision free.

One work uses neural networks to predict collisions [12], but the method is limited to bounding boxes. More recently a contractive autoencoder and a multi-layer perceptron predict collisions from latent space in a probabilistic roadmap [21] setting [31] with very promising results. Our work differs in several ways: a) ClearanceNet uses a simpler neural network architecture, which should lead to faster training and planning, b) we search for the learning hyperparameters, c) we use batching to speed up the planning, and d) we focus on RRTs.

The works that are perhaps most similar to ours are
[8], [9], [10], which use a kernel perceptron and active learning to form a belief model of C-Space. Since these methods are not based on neural networks, they cannot take advantage of the batch processing speedup. Additionally, our method generalizes to object repositioning in the workspace without additional training. CN-RRT takes Fastron-RRT [9] as a foundation and adds whole-edge collision checks, edge building parallelization, adaptive thresholding, and a more flexible repair step.

III. METHODOLOGY

We introduce ClearanceNet, a learned clearance estimator, and its use within a sampling-based motion planning algorithm, termed ClearanceNet-RRT (CN-RRT). ClearanceNet is a neural network that takes a configuration (e.g. a robot pose and object positions) and estimates the minimum clearance between robots and objects in the environment. CN-RRT, a parallel variant of Rapidly-exploring Random Trees (RRTs), then explores the C-Space under the supervision of ClearanceNet and certifies (and possibly repairs) the final motion plan with exact collision checks.

### A. ClearanceNet

**Data Collection:** For a robotic system with C-Space $C$ and free C-Space $C_{\text{free}}$, the data collection process gathers triplets of robot pose $p$, object poses $o$, and clearance $d$ into a dataset $D = \{(p, o, d)|p, o \in C, d \in \mathbb{R}\}$. Robot poses vector $p$ is a joint configuration point for all robots in the workspace for which we are finding a motion plan. The object poses, $o$, is vector of poses of objects in the workspace that can potentially change location between two planning problems, but will not move over the course of one planning problem. For example, a room might contain small tables and shelves (Figure 1) that can be placed in different locations in the scene. The clearance value, $d$, is the minimum distance between the robot and obstacles in the environment or itself; positive values indicate the configuration point is free of collision, i.e., $p \in C_{\text{free}}$, while negative values indicate penetration depth. Given an environment, number of training samples and number of evaluation samples, the data collection algorithm randomly samples a configuration point $p \in C$ and obstacle locations $o$, initializes the environment to that configuration, and computes the minimum clearance $d$ using a traditional geometric method [13]. When the dataset has enough datapoints collected, it is partitioned into training and evaluation datasets.

**ClearanceNet Training:** We train ClearanceNet from the collected dataset. Let $C_\theta$ be a neural network parameterized with weights $\theta$ that takes a configuration point $p$ and object poses $o$ as input and predicts the minimum clearance in the workspace. The neural network consists of two fully connected layers, each followed by a dropout layer, and an output layer that predicts clearance. The training minimizes the mean squared error loss. Finding the right neural network size, dropout rates, learning rate, and minibatch size is not trivial and often time consuming. We automate the search using a large-scale hyperparameter optimization according to Gaussian Process Bandits [30]. This process trains a population of neural nets with different parameters and observes their performance, then selects hyperparameters for the next generation until the hyperparameters are tuned.

**Algorithm 1 CN-RRT**

```plaintext
Input: $p_{\text{init}}, p_{\text{goal}}, o$: Initial and goal poses and obstacles from $C_{\text{free}}$.
Input: $C_\theta$: Clearance estimator.
Input: $d_0^*, d_1^*, t^*$: Initial classifier threshold, second classifier threshold and time to switch.
Input: $N_{\text{parallel}}$: Number of parallel extensions.
Input: $t_{\text{max}}$: Timeout.
Output: $\sigma$: Sequence of poses from $p_{\text{init}}$ to $p_{\text{goal}}$

1: Initialize tree with $p_{\text{init}}$ and threshold $d^*$ as $d_0^*$
2: while $p_{\text{goal}}$ not reached and $t_{\text{elapsed}} < t_{\text{max}}$ do
3: $P_{\text{sample}} \leftarrow$ sample $N_{\text{parallel}}$ points from $C$
4: $P_{\text{expand}} \leftarrow$ nearest node in tree for each $p_s \in P_{\text{sample}}$
5: $P \leftarrow$ discretize path from $p_s$ to $p_g$ for each $(p_s, p_g) \in (P_{\text{expand}}, P_{\text{sample}})$
6: Find clearances for all $P$ in parallel with $C_\theta(P, o)$
7: Traverse each path at first clearance $< d^*$
8: Randomly sample poses at intervals from truncated paths and add them to tree
9: if $t_{\text{elapsed}} > t^*$ then
10: $d^* \leftarrow d_1^* \parallel$ Estimator $C_\theta$ less conservative
11: end if
12: end while
13: $\sigma \leftarrow$ Extract path from tree
14: Validate $\sigma$ and repair with Gradient-based adjustment (Section III-C) and GJK-RRT (Section III-D).
15: return $\sigma$
```

B. CN-RRT: Sampling-based planning with ClearanceNet

The clearance estimator serves as a heuristic to select promising edges in the sampling-based planning setting. To best utilize it, we make five modifications to the Fastron-RRT algorithm [9]. First, taking advantage of neural networks’ ability to efficiently evaluate large batches of data, we present an algorithm that evaluates entire edges for collisions simultaneously. Second, to increase batch size, we sample
Fig. 3. Accuracy for ClearanceNet predicting clearance in different environments. The quadrants represent true positive (top right), false negative (bottom left), false positive (top left), and false negatives (bottom right). False negatives are particularly problematic as they can cause true collisions not to be detected and increase the repair time. False negatives cause CN-RRT to be overly conservative, and the adaptive threshold works around it.

Fig. 4. Inference time over batch size for ClearanceNet over different models. Note both y-axes and the second x-axis are logarithmic. The maximum speedups observed were 132, 261, 287, 193 and 845x respectively for Block, R2D2, Mobile, Ducky and Fetch.

C. Gradient-based Path Repair

Due to the approximate nature of the learned collision checker, it may be necessary to resolve misclassifications. We thus present a gradient-based repair that uses the gradient of ClearanceNet. Recall that ClearanceNet takes a robot pose \( p \) and world state \( o \) and predicts a clearance: \( \hat{C}_p(p, o) \rightarrow d \). By taking the derivative of the output clearance with respect to the input robot state, the result is the direction that moves the robot farther from collision. We scale this by a step size to get a delta: \( \partial \hat{C}_p/\partial p \times \text{step size} = \Delta p_{\text{safety}} \). This derivative is already computed by Tensorflow as part of the gradient and is readily accessible. We use \( \Delta p_{\text{safety}} \) in a novel repair step. First we check all points in the path for collision using GJK. For each point that is in collision, we calculate \( \Delta p_{\text{safety}} \) at the point, find the component perpendicular to the path, and incrementally shift the point towards safety until it is no longer in collision. We then interpolate between any shifted points and repeat the process for these new points. Of course this is not guaranteed to be successful, so after a certain amount of time has passed (we found 1 second to be appropriate), we revert to a GJK-RRT repair detailed in the following section.

D. Classical RRT Repair

To guarantee a collision-free path, we follow an algorithm similar to that presented in [9] to repair any remaining errors. We validate the path with the classic, computationally expensive method GJK. Sections of the path that are in collision are excised. Any dangling, disjointed ends in the path are repaired by building a new, smaller RRT with GJK as the collision checker. Our method differs from that in [9] in that we add a buffer to the in-collision segments and we allow the smaller RRT to connect to any valid point later in the path. The buffer makes it easier to build the small RRT because the start and goal states are not on the cusp of collision. Despite this, the RRT sometimes has trouble and extend toward multiple random points at one time. Third, we introduce adaptive thresholding to control how conservative the algorithm is. Lowering the threshold reduces the number of false negatives (free space predicted to be in collision) but increases the number of false positives. Fourth, we add gradient-based path repair, which is possible because ClearanceNet is differentiable. Fifth, in the repair step we allow the RRT to connect to any valid later point in the trajectory, accelerating repair.

Algorithm I outlines CN-RRT. The tree grows from the start configuration, expanding until it reaches either the goal or the time limit. During each iteration, CN-RRT selects a batch of configuration points and finds their nearest neighbors in the tree. Next, it computes the edges between the random points and their neighbors using a local planner, without checking for collisions. Then ClearanceNet predicts clearances for each configuration point on each edge. This computation is done efficiently in a batch. Then for each edge, starting from the node in the tree, we find the first node where the predicted clearance is smaller than the allowed threshold, i.e predicted collision. The algorithm then adds several samples from the estimated collision-free portion of the edge. Each sample is added to the tree, and all are connected from the expansion node. This step ensures addition of nodes that are not near collision boundaries, although it does increase the size of the tree. Finally, after a given amount of time has passed, if the goal hasn’t been reached, the algorithm lowers the classification threshold allowing more misclassified collisions, but also more edges added.
TABLE II
SUMMARY OF PATH CHARACTERISTICS AND PATH-BUILDING STATISTICS FOR CNN-RRT AND BASELINE METHODS AVERAGED OVER 500 QUERIES (300 FOR MOBILE; 50 FOR FETCH). THE %Δ COLUMNS SHOW PERCENT INCREASE OR DECREASE AS COMPARED TO CN-RRT. THE PATH DURATION COLUMN GROUP ONLY CONSIDERS SUCCESSFUL QUERIES, WHICH BIASES IT IN FAVOR OF METHODS THAT SOLVE ONLY THE SHORTER, EASIER PROBLEMS.

| Environment | Method     | Calculation Time (s) | Path Duration (s) | Collision Checks | Heuristic Collision Checks |
|-------------|------------|----------------------|-------------------|------------------|---------------------------|
|             |            | µ | σ² | µ | σ² | µ | σ² | µ | σ² |
| Block       | CN-RRT     | 2.36 | +0 | 6.25 | 1.02 | -0 | 0.41 | 14777 | 56369 | 20686 | 129973 |
|             | CN-RRT-Grad| 1.71 | -37 | 5.12 | 0.97 | -5 | 0.40 | 7413 | 41915 | 19033 | 85870 |
|             | GJK-RRT    | 2.74 | +14 | 6.66 | 1.21 | +16 | 0.50 | 24493 | 67788 | 0 | 0 |
|             | Fastron-RRT| 15.73 | +85 | 13.91 | 1.17 | +13 | 0.48 | 11754 | 48716 | 43603 | 43398 |
| R2D2        | CN-RRT     | 1.96 | +0 | 4.59 | 0.98 | +0 | 0.44 | 5423 | 22850 | 43141 | 171283 |
|             | CN-RRT-Grad| 1.37 | -43 | 3.42 | 0.93 | -5 | 0.42 | 483 | 7206 | 39615 | 122176 |
|             | GJK-RRT    | 2.83 | +31 | 5.24 | 1.26 | +22 | 0.52 | 14513 | 30968 | 0 | 0 |
|             | Fastron-RRT| 3.37 | +42 | 5.00 | 1.17 | +17 | 0.50 | 12918 | 34324 | 3761 | 3371 |
| Ducky       | CN-RRT     | 4.77 | +0 | 9.04 | 0.94 | -0 | 0.43 | 26729 | 59188 | 6832 | 11312 |
|             | CN-RRT-Grad| 3.68 | -30 | 8.53 | 0.67 | -41 | 0.34 | 18783 | 52247 | 6787 | 14028 |
|             | GJK-RRT    | 4.86 | +2 | 9.19 | 1.05 | +11 | 0.43 | 25675 | 51259 | 0 | 0 |
|             | Fastron-RRT| 31.30 | +85 | 2.79 | 0.74 | -27 | 0.31 | 58 | 699 | 47427 | 5222 |
| Mobile      | CN-RRT     | 91.64 | +0 | 67.32 | 2.92 | +0 | 1.34 | 30621 | 26538 | 25936 | 523550 |
|             | GJK-RRT    | 89.75 | -2 | 73.41 | 3.22 | +9 | 1.56 | 41110 | 337983 | 0 | 0 |
|             | Fastron-RRT| 180.42 | +49 | 0.29 | 2 | +0 | 0 | 481407 | 83311 |
| Fetch       | CN-RRT     | 416.83 | +0 | 417.50 | 35.12 | +0 | 21.11 | 436720 | 435556 | 120697 | 149119 |
|             | GJK-RRT    | 599.18 | +30 | 426.40 | 46.95 | +25 | 24.48 | 617493 | 439295 | 0 | 0 |
|             | Fastron-RRT| 923.13 | +55 | 266.72 | 8.56 | -310 | 2.07 | 24801 | 163753 | 1268222 | 423641 |

Connecting to a given goal point, so we make the task easier by probabilistically sampling all points later in the path that have not been excised. This both gives the RRT more possible goals to hit and also sometimes allows us to bridge multiple bad segments with a single repair RRT.

IV. RESULTS

Environment Setup: We evaluate ClearanceNet and CN-RRT (Algorithm 1) on five environments depicted in Figure 2. Two environments (Block and R2D2) contain two 7-DoF Kuka arms with their bases fixed to a planar surface. The first environment also contains a floating block between the arms (Figure 2a), while the second contains a model of R2D2 (Figure 2b). The third environment (Ducky) contains one fixed-base 7-DoF Kuka arm and three objects: a rubber duck, R2D2, and a cube (Figure 2c). The objects are selected to have different geometric properties, from the uniform block to the duck with smooth curved surfaces. The environment is parameterized with the locations of the origins of the objects. The object locations are varied for each motion planning problem. Thus motion planning in this environment is 7-DoF, but ClearanceNet input length is 16; 7 for the robot pose, and 3 for each obstacle in the scene. The fourth environment consists of three mobile manipulators: Kuka arms attached to moving bases with 50 cm sides (Figure 2d). In this environment, the collision checks between the three mobile manipulators are done as three pair-wise checks. Motion planning is 30-DoF, and ClearanceNet input length is 17 for each pair-wise check; 7 × 2 for arm joints, 1 for base separation and 2 for base orientations. The fifth environment contains a Fetch robot, two sets of shelves and two cubes in a 2m × 2m square. The shelves can slide along one edge of the floor and the cubes can be at any position on the floor. Motion planning is thus 11-DoF (7 arm joints, 1 torso lift, 2 base position, 1 base orientation), and ClearanceNet input length is 17; 11 for Fetch, 1 for each set of shelves and 2 for each cube.

A. ClearanceNet Results

Training: We generate datasets of 1,000,000 training samples and 10,000 evaluation samples for each environment using Bullet Physics Simulation [7], which computes the separation distances using the Gilbert-Johnson-Keerthi (GJK) distance algorithm [13]. The data collection takes 17-130 minutes per environment. The neural networks have layers with 1400 neurons each, for a total of approximately 2 million parameters. We train using Adam optimizer [22] with mean squared error loss.

Hyperparameters: We use Vizier [14] with Gaussian Bandits [30] to tune the hyperparameters. We search for hyperparameters using a total population of 1000 models, running 100 models at the time. Single network training completes in 1-2 hours on average, while the hyperparameter search completes within 24 hours. We find that one set of optimized hyperparameters performs well across the environments and robot setups; thus we perform the hyperparameter tuning only once. The parameters found are dropout rate = 0.01, learning rate = 0.000174950, batch size = 191. We note that the dropout rate is surprisingly low. The batch size is not a power of two due to a setup quirk.

Accuracy: Table I summarizes ClearanceNet’s accuracy across the environments, shown in more detail in Figure 3. Intuitively, high false negative rates are expected to be particularly harmful, as they may lead to trajectories with collisions being classified as free space. However, in practice false positives are also problematic because they lead to conservative planners that find no paths at all. A balance of false positives and false negatives is best, as achieved for all environments except Fetch.

Inference Speed: Figure 4 shows ClearanceNet’s inference-time speed over batch size. Note that the y-axes are logarithmic. ClearanceNet becomes faster with larger
Fig. 5. Success rate (left) and time per phase (right) across environments. The line plots show fraction of problems solved over time. Fastron does not appear in Mobile because it failed to solve any problems. The bar graphs show average time spent by each method in each phase. Fastron is included when its scale is comparable to the other methods. Failure to solve the problem is counted as the maximum time allotted.

### B. CN-RRT Results

We compare CN-RRT to two baselines: RRT with GJK collision checking, and RRT with Fastron [9] collision checking. ClearanceNet is implemented with TensorFlow [1] using the Python interface. Bullet is implemented in C++, and we use the Python interface PyBullet. Fastron was implemented in Python using Numpy. All experiments ran on an Nvidia Tesla V100 GPU with 16 Gbs of RAM.

We evaluate all methods on test sets with fixed start and goal configurations. For the first three environments (Block, R2D2 and Ducky) we test 100 problems per environment, with 5 random seeds per problem and a 30 second timeout per seed. For Mobile: 100 problems, 3 seeds, 180 seconds. For Fetch: 10 problems, 5 seeds, 1,000 seconds. For CN-RRT and Fastron-RRT, the computation time includes validation and repair.

**CN-RRT success rate and planning speed:** CN-RRT is more likely to find a solution for a query than the comparison methods most of the time (Figure 5). In the Ducky environment the performance is comparable to GJK. In the Mobile environment, which contains more robots in the evaluation than during training, GJK solves more easier solutions faster. However, as the queries become more difficult CN-RRT finds solutions that the comparison methods could not. In all other environments CN-RRT consistently solves more queries faster (Table II).

**CN-RRT time allocation:** CN-RRT solves a single query faster than the comparison methods for all environments except Mobile (Figure 5). In addition, the average repair time for CN-RRT is shorter than the average build time for GJK-RRT, demonstrating that repairing a trajectory is quicker than building one from scratch.

**CN-RRT paths optimality:** Next, we examine the quality of the paths that CN-RRT finds. In every environment CN-RRT paths are shorter on average than GJK-RRT (Table II). Fastron-RRT has shorter paths in Ducky and Fetch environments, but this is a result of Fastron-RRT only solving a small subset of easier problems.

### C. Gradient-based Repair Results

Gradient-based repair is applicable only to degrees of freedom that are holonomic. In the case of our environments, this means Block, R2D2 and Ducky. It would work for Fetch’s arm but not for its base, which cannot translate sideways. Mobile is a special case because the collision checks are done pair-wise. For Block, R2D2 and Ducky, there are two parameters we can tune: the size of each step toward safety and the number of extra steps to take once a point in $C_{free}$ has been found. We run a grid of experiments per-environment on a training set of motion planning problems to determine the best values for these parameters. For Block, step size is 0.15 and extra steps is 0. For R2D2 and Ducky, step size is 0.05 and extra steps is 3.

For all three environments, gradient-based repair is an improvement over GJK-based repair in terms of both average calculation time and average path duration (Figure 5). This is true even though initial build time increases as a result of
using Tensorflow eager mode. We note the validation time appears to decrease from CN-RRT to CN-RRT-Grad, but this is misleading because most validation instead occurs in the “Shift” phase.

D. On robot experiments

We validate CN-RRT by taking a path it found and executing it on a real Fetch robot [27]. The start pose is randomly generated, and the end pose is selected to be especially challenging: it requires contact with one of the shelves and only 1 mm clearance between the shelf wall and gripper (see Figure 1). For this query, CN-RRT’s planning time is 47 seconds, while the comparison methods fail to find a solution. We repeat the on-robot experiment three times, and all three runs are successful.

V. DISCUSSION

Run-time analysis: The cost of motion planning with CN-RRT includes the cost of data collection, clearance estimator training, and motion planning. The data collection and training are one-time per environment costs, with data collection requiring about 30 minutes, and training about 60 minutes for a total overhead of 90 minutes. Given that the best case average query savings (from the Fetch environment) is 182 seconds, that means it takes 29 queries to save time in the Fetch environment.

Generalization: The environments that we choose demonstrate generalization across multiple dimensions. Block and R2D2 show hyperparameter generalization across different obstacles, obstacle locations and robot locations. Ducky shows generalization across different arrangements of obstacles, with no retraining required. And Mobile and Fetch show that this method is applicable beyond robotic arms.

Limitations and future work: One shortcoming of the method is that all objects in the environment must be known beforehand. There is currently no way to handle, for example, a person walking into the environment. An area for future work is addressing this shortcoming by modifying the input and architecture of the neural network.

VI. CONCLUSIONS

We have presented ClearanceNet, a deep network for estimating minimum clearance, and CN-RRT, an RRT based algorithm that parallelizes calls to the collision checker and uses adaptive thresholding to make problems easier to solve. Evaluated on five environments, the proposed method produces shorter paths more quickly. On-robot experiments demonstrate the method’s applicability to real robots.

VII. ACKNOWLEDGEMENTS

The authors thank Hao-Tien Lewis Chiang for helpful discussions, Laura Downs for virtual asset creation, and Krista Reymann for coordination.
REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, J. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] N. M. Amato and L. K. Dale. Probabilistic roadmap methods are embarrassingly parallel. In Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C), volume 1, pages 688–694 vol.1, May 1999.

[3] J. Bialkowski, S. Karaman, and E. Frazzoli. Massively parallelizing the rrt and the rrt. In 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 3513–3518. IEEE, 2011.

[4] J. Bialkowski, M. Otte, S. Karaman, and E. Frazzoli. Efficient collision checking in sampling-based motion planning via safety certificates. The International Journal of Robotics Research, 35(7):767–796, 2016.

[5] L. Chiang, A. Faust, S. Sugaya, and L. Tapia. Fast swept volume estimation with deep learning. WAFR, 2018.

[6] S. Choudhury, M. Bhardwaj, S. Arora, A. Kapoor, G. Ranade, S. Scherer, and D. Dey. Data-driven planning via imitation learning. The International Journal of Robotics Research, 37(13-14):1632–1672, 2018.

[7] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org 2016–2019.

[8] N. Das, N. Gupta, and M. Yip. Fastron: An online learning-based model and active learning strategy for proxy collision detection. In 1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings, pages 496–504, 2017.

[9] N. Das and M. Yip. Learning-based proxy collision detection for robot motion planning applications. CoRR, abs/1902.08164, 2019.

[10] N. Das and M. Yip. Learning-Based Proxy Collision Detection for Robot Motion Planning Applications. feb 2019.

[11] M. Elbanhawi and M. Simic. Sampling-based robot motion planning: A review. IEEE Access, 2:56–77, 2014.

[12] I. Garcia, J. D. Martin-Guerrero, E. Soria-Olivas, R. J. Martinez, S. Rueda, and R. Magdalena. A neural network approach for real-time collision detection. In IEEE International Conference on Systems, Man and Cybernetics, volume 5, pages 5 pp. vol.5–, Oct 2002.

[13] E. Gilbert, D. W. Johnson, and S. Keerthi. A fast procedure for computing the distance between objects in three-dimensional space. Robotics and Automation, IEEE Journal of, 4:193 – 203, 05 1988.

[14] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley. Google vizier: A service for black-box optimization. In Proc. of ACM International Conference on Knowledge Discovery and Data Mining, pages 1487–1495. ACM, 2017.

[15] Y. J. Heo, D. Kim, W. Lee, H. Kim, J. Park, and W. K. Chung. Collision detection for industrial collaborative robots: A deep learning approach. IEEE Robotics and Automation Letters, 4(2):740–746, apr 2019.

[16] K. Hornik. Approximation capabilities of multilayer feedforward networks. Neural Networks, 4(2):251 – 257, 1991.

[17] J. Huh and D. D. Lee. Learning high-dimensional mixture models for fast collision detection in rapidly-exploring random trees. In Proc. IEEE Int. Conf. Robot. Autom. (ICRA), pages 63–69. IEEE, 2016.

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

[19] B. Ichter and M. Pavone. Robot motion planning in learned latent spaces. IEEE Robotics and Automation Letters, 2019.

[20] G. Kahn, A. Villalba, V. Pong, P. Abbeel, and S. Levine. Uncertainty-aware reinforcement learning for collision avoidance, 2017.

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

[22] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014.

[23] S. Kumar, S. Choudhary, and S. Srinivasa. Estimating configuration space belief from collision checks for motion planning. CoRR, abs/1901.07646, 2019.

[24] S. M. LaValle. Planning Algorithms. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[25] Y. Li and K. E. Bekris. Learning approximate cost-to-go metrics to improve sampling-based motion planning. In 2011 IEEE International Conference on Robotics and Automation, pages 4196–4201. IEEE, 2011.

[26] A. Loquercio, A. I. Maqueda, C. R. Del-Blanco, and D. Scaramuzza. DroNet: Learning to Fly by Driving. IEEE Robotics and Automation Letters, 3(2):1088–1095, apr 2018.

[27] D. K. E. D. Melonee Wise, Michael Ferguson and D. Dymesich. Fetch & freight: Standard platforms for service robot applications. In Workshop on Autonomous Mobile Service Robots held at the 2016 International Joint Conference on Artificial Intelligence, 2016.

[28] J. Pan and D. Manocha. Fast probabilistic collision checking for sampling-based motion planning using locality-sensitive hashing. The International Journal of Robotics Research, 35(12):1477–1496, 2016.

[29] J. Pan, X. Zhang, and D. Manocha. Efficient penetration depth approximation using active learning. ACM Transactions on Graphics, 2013.

[30] N. Srinivas, A. Krause, S. Kakade, and M. Seeger. Information-theoretic regret bounds for gaussian process optimization in the bandit setting. IEEE Transactions on Information Theory, 58(5):3250–3265, May 2012.

[31] T. N. Tran, J. Denny, and C. Ekenna. Predicting collision detection with a two-layer neural network. 2019.