Neural Collision Clearance Estimator for Batched Motion Planning

J. Chase Kew[1], Brian Ichter[1], Maryam Bandari[2], Tsang-Wei Edward Lee[1], and Aleksandra Faust[1]

[1] Robotics at Google, Mountain View, CA, USA,
{jkew,ichter,tsangwei,faust}@google.com
[2] X, Mountain View, CA, USA, maryamb@x.team

Abstract. We present a neural network collision checking heuristic, ClearanceNet, and a planning algorithm, CN-RRT. ClearanceNet learns to predict separation distance (minimum distance between robot and workspace) with respect to a workspace. CN-RRT then efficiently computes a motion plan by leveraging three key features of ClearanceNet. First, CN-RRT explores the space by expanding multiple nodes at the same time, processing batches of thousands of collision checks. Second, CN-RRT adaptively relaxes its clearance requirements for more difficult problems. Third, to repair errors, CN-RRT shifts its nodes in the direction of ClearanceNet's gradient and repairs any residual errors with a traditional RRT, thus maintaining theoretical probabilistic completeness guarantees. In configuration spaces with up to 30 degrees of freedom, ClearanceNet achieves 845x speedup over traditional collision detection methods, while CN-RRT accelerates motion planning by up to 42% over a baseline and finds paths up to 36% more efficient. Experiments on an 11 degree of freedom robot in a cluttered environment confirm the method's feasibility on real robots.

1 Introduction

Modern robots work in complex workspaces such as factory lines and warehouses. They shelve inventory (Fig. 1), inspect parts (Fig. 2b), and perform collaborative tasks (Fig. 2d). Within its lifetime a robot can complete hundreds of thousands of tasks. In factories and warehouses the workspaces change in structured ways: objects are displaced as the robot stocks the shelves, and free space depends on other robots' poses. However, classic motion planning takes little advantage of this structure in changing environments and typically plans every path from scratch. In this work, we present a learning-based method that leverages prior experience to quickly propose paths and pass them to a computationally expensive geometry-based planner for validation and repair.

Motion planning is the problem of finding a feasible path between start and goal poses in the robot's workspace. Often, motion planning problems are solved in configuration space ($\mathcal{C}$), the set of possible robot poses [15] within a static workspace. As the dimensionality of $\mathcal{C}$ increases, exactly solving the planning

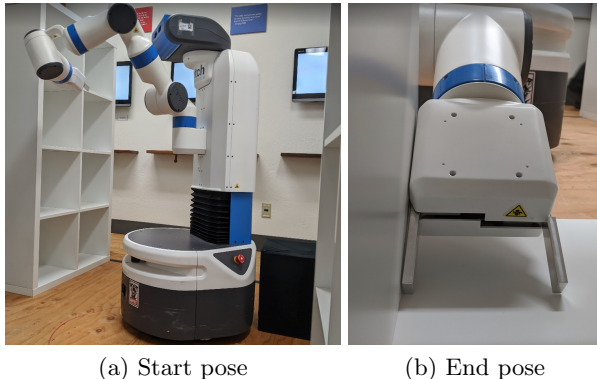(a) Start pose   (b) End pose

Fig. 1. Real-world validation with Fetch robot reaching into a shelf.

problem quickly becomes intractable. Sampling-based planning addresses this issue by forming an approximate, implicit representation of $\mathcal{C}$ through a set of probing random samples, connected locally by querying a collision checker.

Classical approaches to collision checking rely on computing complicated geometries. Though there exists a rich literature on efficient computation [17, 11], 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 by certifying regions to be collision free [3] or by using probabilistic collision checks [14] to provide a quick, approximate belief about which regions are in collision. Regardless, collision checking remains the primary bottleneck in sampling-based motion planning, consuming up to 90% of total computation time [8].

We address the collision checking bottleneck and accelerate sampling-based planning with three key insights. First, separation distance is a function of only the poses of all mobile bodies in an environment. This leads to the idea of learning to predict separation distance (clearance) from examples, without the robot explicitly knowing either its own geometry or the obstacles' during planning. This works because deep neural networks are universal approximators that with enough data can learn any continuous function and because clearance is a Lipschitz continuous function [15]. We therefore train a deep neural network, ClearanceNet, to predict collision clearance for robots and workspaces with many degrees of freedom (DoFs), using geometry for training but not for inference. The second insight is that both sampling-based motion planning and neural network inference are embarrassingly parallelizable [1, 2]. Leveraging neural networks' efficiency at processing large batches, we collision check entire edges together and expand from many RRT nodes in parallel. This batch processing leads to a heuristic collision check two orders of magnitude faster than optimized geometric methods on a single GPU. The third insight is that often it is easier to verify and repair a partially-correct trajectory than to plan from scratch. This is relevant because, although fast, the learned collision heuristic is approximate and makes mistakes. Thus, building on [7], we verify and repair motion plans
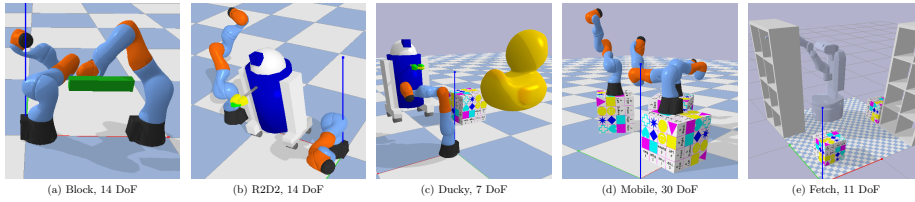
(a) Block, 14 DoF   (b) R2D2, 14 DoF   (c) Ducky, 7 DoF   (d) Mobile, 30 DoF   (e) Fetch, 11 DoF

Fig. 2. Environments: a), b) Static workspace with fixed robot bases. c) Parameterized workspace - robot is fixed while obstacle placement varies. d) Mobile manipulators. Model trained for two mobile manipulators and used to plan for three without additional training. e) Fetch robot with 7 DoF arm, torso lift and base position. Each set of shelves has one DoF and cubes have 2 DoFs each.

using the gradient of the neural network to shift poses towards higher clearance, then repair remaining misclassified poses with traditional RRT [16].

Using these insights, this paper makes three contributions. 1) *ClearanceNet*, a neural network that estimates minimum separation distance, conditioned on joint robot-workspace pose. ClearanceNet is trained using large-scale hyperparameter tuning that simultaneously searches for the appropriate training hyperparameters and trains the clearance estimator. We also provide the training hyperparameters, which were consistent across environments and robots. 2) *CN-RRT*, an adaptive and batched extension of the Fastron-RRT algorithm [7], which leverages the efficiency of neural network inference when processing large batches of data (up to 60 edges and 5,000 collision checks). CN-RRT adaptively reduces the difficulty of the motion planning problem and includes a repair step that attempts to move the path away from obstacles using ClearanceNet's gradient. 3) A configurable workspace, extended configuration space and in-depth analysis of CN-RRT trade offs.

ClearanceNet and CN-RRT are evaluated on five environments (Fig. 2) with convex and concave obstacles. Robots range from 7 to 30 DoFs, including multiple arms with fixed bases, mobile manipulators, and a full bodied Fetch robot. We compare the presented method with two baselines: a traditional C++ optimized geometric collision checker (Gilbert-Johnson-Keerthi or GJK [9]) implemented in PyBullet [6], and Fastron [7], a learned collision checker with a kernel perceptron. ClearanceNet performs collision checking up to 845x faster than GJK in sufficiently large batches, and achieves accuracy of 91-96%. CN-RRT both completes planning faster (up to 42%) and produces up to 36% shorter paths.

2 Related Work

Several previous works used machine learning to offset the cost of collision checking. Some methods learn heuristics for promising paths [4], learn a distribution of promising regions [13], or evaluate only the most promising edges [5] based on predicted energy costs [23]. Our approach, which learns to accelerate the actual collision checking, can be used in conjunction with these approaches.

Other works use learning to accelerate collision checking itself by approximating $\mathcal{C}$ with Gaussian mixture models [12], or by classifying collisions based on

nearby collision checks [14, 18]. Another work trains a support vector machine offline and actively adapts it online to predict the penetration depth of rigid bodies [19]. Our method, however, focuses specifically on learning to classify collisions in the configuration space and ultimately certifies path validity.

In [21] a contractive autoencoder and multi-layer perceptron predict collisions from latent space in a probabilistic roadmap. Our work differs in that: 1) batching speeds up planning, 2) a repair step guarantees correctness and 3) inputs are configuration points instead of occupancy grids, allowing us to expand the workspace volume without changing the neural network architecture and to support obstacles with arbitrary geometry, beyond axis-aligned boxes. The work most similar to ours is Fastron [7], which uses a kernel perceptron and active learning to form a belief model of $\mathcal{C}$. Because Fastron is not based on a neural network, it cannot take advantage of the batch processing speedup. Additionally, our method generalizes to object repositioning in the workspace without further training. CN-RRT takes Fastron-RRT [7] as a foundation and adds whole-edge collision checks, edge building parallelization, adaptive thresholding, and a flexible repair step to guarantee probabilistic completeness.

New simulation environments are emerging for path planning that allow users to programmatically create new workspaces by randomly placing a fixed number of objects within a static environment [22]. Here we consider a similar setting, where the workspace consists of fixed obstacles and obstacles that may change positions between two motion planning problems.

3   Problem Statement

In this paper we use relaxed assumptions about robot workspace and configuration space. To that end, this section introduces key definitions for workspace configuration space, extended configuration space and our problem definition.

Definition 1. *Workspace configuration space.* $\mathcal{C}_w \subset \mathbb{R}^{d_w}$ *is the set of all possible workspace configurations for* $d_w \geq 0$. *Vector* $\boldsymbol{w} \in \mathcal{C}_w$ *is a* ***workspace configuration*** *and scalar* d_w *is the* ***workspace degrees of freedom***. *We denote with* $\mathcal{W}(\boldsymbol{w})$ *a workspace with configuration* $\boldsymbol{w}$.

Intuitively, a workspace configuration space corresponds to a set of related workspaces where large portions are fixed, but there are several objects that can move between planning problems. In a typical room, for example, the floor, ceiling, tables and cabinets remain static, while the locations of chairs may change. In the special case $d_w = 0$, $\mathcal{C}_w = \varnothing$, and there is only one possible workspace corresponding to the classic static workspace $\mathcal{W}$.

Definition 2. *Extended configuration space.* $\mathcal{C}_e$ *is the set of all possible robot and workspace configurations,* $\mathcal{C}_e = \mathcal{C} \times \mathcal{C}_w \subset \mathbb{R}^{d_e}$, $d_e = d_r + d_w$, *where* $\mathcal{C} \subset \mathbb{R}^{d_r}$ *is the set of all possible robot poses* $\boldsymbol{p}_r$. *A configuration space for a fixed workspace configuration* $\boldsymbol{w}$ *is denoted by* $\mathcal{C}_e(\boldsymbol{w})$.

The extended configuration space, a configuration space generalization, contains d_r robot controllable DoFs, and d_w DoFs not controllable by the robot. Robot

pose $\mathbf{p}_r$ is a joint configuration for all robots in the workspace for which we are finding a motion plan, while the workspace may contain objects that change positions between motion planning instances. In the special case $d_w = 0$ all DoFs of the extended configuration space are controllable by the robot, and $\mathcal{C}_e = \mathcal{C}$.

Definition 3. *Collision checking in $\mathcal{C}_e$. A configuration $\boldsymbol{p} = (\boldsymbol{p}_r, \boldsymbol{w})$ is valid iff robot configuration $\boldsymbol{p}_r$ is valid in workspace $\mathcal{W}(\boldsymbol{w})$. **Extended free configuration space**, $\mathcal{C}_{ef} \subset \mathcal{C}_e$ is the set of all valid configurations $\mathcal{C}_{ef} = \{\boldsymbol{p} = (\boldsymbol{p}_r, \boldsymbol{w}) \mid isValid(\boldsymbol{p}_r, \mathcal{W}(\boldsymbol{w})), \boldsymbol{p}_r \in \mathcal{C}, \boldsymbol{w} \in \mathcal{C}_w\}$. The free configuration space for a fixed $\boldsymbol{w}$ is denoted by $\mathcal{C}_f(\boldsymbol{w})$.*

In other words, a configuration is valid in joint robot-workspace configuration space if the robot pose is valid in this particular instance of the workspace.

Definition 4. *Path planning in extended configuration space. Given extended configuration space $\mathcal{C}_e$, workspace configuration $\boldsymbol{w}$, start and end robot poses, $\boldsymbol{p}_{rs}, \boldsymbol{p}_{rg}$, find a valid path $\mathcal{P} = [\boldsymbol{p}_{r0}, \cdots, \boldsymbol{p}_{rn}]$, such that the path starts in the start robot pose, ends in the end robot pose, and each point in the path is valid: $\boldsymbol{p}_{r0} = \boldsymbol{p}_{rs}, \boldsymbol{p}_{rn} = \boldsymbol{p}_{rg}, \boldsymbol{p}_{ri} \in \mathcal{C}_f(\boldsymbol{w})$ for $i = 1, \cdots, n$.*

Definition 5. *Data batch. Let $\boldsymbol{p} \in \mathbb{R}^{d_e}$ be a vector, $\mathcal{B} = [\boldsymbol{p}_1, \cdots, \boldsymbol{p}_n]^\intercal$ be a $d_e \times n$ matrix of batched data, and $C_{\hat{\boldsymbol{\theta}}} : \mathbb{R}^{dofs} \to \mathbb{R}$ be a neural network with d_e inputs and one output. Batched inference $\boldsymbol{C}_{\hat{\boldsymbol{\theta}}}$ maps a batched data matrix $\mathcal{B}$ into an n-dimensional vector.*

Deep neural networks are optimized to efficiently process large batches of data using low-level parallelization of matrix multiplication. Leveraging this property of neural networks is key in our algorithm's multi-directional RRT growth.

4   Methodology

We introduce ClearanceNet, CN: $\mathcal{C}_e \to \mathbb{R}$, a learned clearance estimator, and its use within a sampling-based motion planning algorithm, CN-RRT. ClearanceNet is a neural network that takes a configuration (e.g. a robot pose and workspace configuration) and estimates the minimum clearance between robot and workspace. CN-RRT, a batched variant of Rapidly-exploring Random Trees (RRTs), then explores $\mathcal{C}$ under the supervision of ClearanceNet and certifies (and possibly repairs) the final motion plan with exact collision checks.

4.1   ClearanceNet

Data Collection: The data collection process samples a robot pose $\mathbf{p}_r$ and workspace configuration $\mathbf{w}$ from the extended configuration space. It sets the workspace to $\mathcal{W}(\mathbf{w})$ and robot to $\mathbf{p}_r$ and computes the clearance d, the minimum distance between the robot and the workspace or itself, using a geometric method [9]. Positive values indicate the configuration point is free of collision, i.e. $\mathbf{p}_r \in \mathcal{C}_{ef}(\mathbf{w})$, while negative values indicate penetration depth. The triples are then added to a dataset: $\mathcal{D} = \{(\mathbf{p}_r, \mathbf{w}, d) \mid (\mathbf{p}_r, \mathbf{w}) \sim \mathcal{C}_e, d \in \mathbb{R}\}$. When the dataset is large enough, it is partitioned into training and evaluation sets.

Algorithm 1 CN-RRT

Input: Initial and goal robot poses: $\mathbf{p}_{rs}, \mathbf{p}_{rg}$ and a workspace configuration: $\mathbf{w}$.
Input: $\mathbf{C}_{\hat{\boldsymbol{\theta}}}$: Trained clearance estimator.
Input: $\boldsymbol{d}^*, \boldsymbol{\Delta t}$: Monotonically decreasing vectors of CN thresholds and times to switch.
Input: N_{ext}: Number of simultaneous extensions.
Input: $t_{\max}$: Timeout.
Output: $\mathcal{P}$: Sequence of robot poses from $\mathbf{p}_{rs}$ to $\mathbf{p}_{rg}$.
1: Initialize tree $\mathcal{T}$ with $\mathbf{p}_{rs}$ and current threshold index $j = 1$.
2: **while** $\mathbf{p}_{rg}$ not reached and $t_{\text{elapsed}} < t_{\max}$ **do**
3:     $P_{\text{sample}} \leftarrow [\mathbf{p}_r^{(1,s)}, \cdots, \mathbf{p}_r^{(N_{\text{ext}},s)} | \mathbf{p}_r^{(i,s)} \sim \mathcal{C}_e(\mathbf{w})]$. Sample N_{ext} points from $\mathcal{C}_e(\mathbf{w})$.
4:     $P_{\text{expand}} \leftarrow [\mathbf{p}_r^{(1,e)}, \cdots, \mathbf{p}_r^{(N_{\text{ext}},e)} | \mathbf{p}_r^{(i,e)} :$ Nearest in $\mathcal{T}$ to $\mathbf{p}_r^{(i,s)}]$. Nodes to expand.
5:     $P \leftarrow$ Discretized paths from $\mathbf{p}_r^{(i,e)}$ to $\mathbf{p}_r^{(i,s)}$ for each $i = 1, \cdots, N_{\text{ext}}$
6:     Find clearances $\mathbf{c} \leftarrow \mathbf{C}_{\hat{\boldsymbol{\theta}}}(\mathcal{B})$, for the batch of points, $\mathcal{B} = \{(\mathbf{p}_r, \mathbf{w}) | \mathbf{p}_r \in P\}$.
7:     Truncate each path at first clearance $< \boldsymbol{d}^*[j]$
8:     Randomly sample poses from truncated paths and add them to tree $\mathcal{T}$.
9:     **if** $t_{\text{elapsed}} > \boldsymbol{\Delta t}[j]$ **then**
10:        $j \leftarrow \min(j + 1, \text{length}(\boldsymbol{d}^*))$. Estimator $C_{\hat{\boldsymbol{\theta}}}$ less conservative.
11:    **end if**
12: **end while**
13: $\mathcal{P} \leftarrow$ **if** $(\mathbf{p}_{rg} \in \mathcal{T})$ **then** Extract path from tree $\mathcal{T}$ **else** $[\mathbf{p}_{rs}, \mathbf{p}_{rg}]$.
14: **return** Algorithm 2 $(\mathcal{P})$ /* Validate, and repair if necessary, path $\mathcal{P}$. */

ClearanceNet Training: We train ClearanceNet from the collected dataset $\mathcal{D}$. Let $C_{\boldsymbol{\theta}} : \mathcal{C}_e \to \mathbb{R}$ be a neural network parameterized with weights $\boldsymbol{\theta}$ that takes a robot and a workspace configuration point $(\mathbf{p}_r, \mathbf{w}) \in \mathcal{C}_e$ as input and predicts the minimum clearance d in the workspace $\mathcal{W}(\mathbf{w})$. 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, $\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} N^{-1} \sum_{i=1}^{N} \|d^{(i)} - C_{\boldsymbol{\theta}}(\mathbf{p}_r^{(i)}, \mathbf{w}^{(i)})\|^2$. 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 [20]. 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. A trained ClearanceNet is denoted by $C_{\hat{\boldsymbol{\theta}}}$.

Collision classifier definition: After the clearance estimator $C_{\hat{\boldsymbol{\theta}}}$ is trained, we make a collision classifier by selecting a clearance threshold d^*. A pose $\mathbf{p}_r$ is classified as in collision if $C_{\boldsymbol{\theta}}(\mathbf{p}_r, \mathbf{w}) < d^*$, i.e. the predicted clearance is less than the threshold. We can vary d^* to obtain a stricter or more permissive classifier.

4.2   CN-RRT: Sampling-based planning with ClearanceNet

ClearanceNet serves as a heuristic to select promising edges in a sampling-based planning setting. To best utilize it, we make three modifications to the Fastron-RRT algorithm [7]. First, taking advantage of neural networks' ability to efficiently evaluate large batches of data, we present an algorithm that evaluates sev-

eral entire edges for collisions in one large batch. Second, we introduce adaptive thresholding to control how conservative the algorithm is. Lowering the threshold reduces false positives (free space predicted to be in collision) but increases false negatives. Last, we add a gradient-based path repair, which is possible because ClearanceNet is differentiable in configuration space. Intuitively, during repair we move the path away from obstacles in the direction of the gradient.

Batch Node Expansion: Algorithm 1 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 (Line 3) and finds their nearest neighbors in the tree (Line 4). Next, it computes the edges between the random points and their neighbors using a local planner, *without* checking for collisions (Line 5). Then ClearanceNet predicts clearances for the entire batch that contains configuration points for all edges (Line 6). Next, for each edge, CN-RRT finds the first node where the predicted clearance is smaller than the allowed threshold, i.e predicted collision (Line 7). The algorithm then adds several samples from the estimated collision-free portion of the edge (Line 8). Each sample is added to the tree, and all are connected from the expansion node (Line 8). This step ensures addition of nodes that are not near collision boundaries, although it does increase the size of the tree.

Clearance Threshold Adaptation: CN-RRT has an adaptive clearance threshold d^* to make the algorithm more flexible (note that d^* controls the precision of the classifier). An overly conservative classifier will unnecessarily discard collision-free samples, but most samples classified as collision free will be truly collision free. Many easy motion planning problems (less cluttered workspace, start and end poses nearby, etc.) can be solved even with an overly conservative classifier, and these will require little or no repair, but other problems require less conservative solutions. Because we do not know ahead of time what threshold is appropriate for a given problem, we give CN-RRT a list of thresholds and time limits as input. If it finds no solution within a given time limit (Line 9), the algorithm decreases the threshold (Line 10) to relax the classifier. This step ensures that in subsequent iterations more nodes are added to the tree, although they may not be truly collision free.

Gradient-based Path Repair: Because the learned collision checker is approximate, CN-RRT may need to resolve misclassifications. It does so with a repair step that uses the gradient of ClearanceNet. Recall that ClearanceNet is a differentiable scalar function of robot pose $\mathbf{p}_r$ and workspace state $\mathbf{w}$ that predicts a clearance, $C_{\hat{\boldsymbol{\theta}}}(\mathbf{p}_r, \mathbf{w}) \to d$. This gradient is automatically computed to train the network. This means that for a robot pose in collision, following the positive gradient of the clearance with respect to robot pose can increase the clearance; initially reducing the amount of collision and eventually finding a collision-free point. Let $\mathbf{p}^{(i)}, \mathbf{p}^{(i+1)} \in \mathcal{P}$ be two consecutive configurations points in the proposed path. Let $\mathbf{g}^{(i)} = \nabla C_{\hat{\boldsymbol{\theta}}}(\mathbf{p}^{(i)})$ be a gradient of the ClearanceNet w.r.t. the point, and $\Delta\mathbf{p}^{(i)} = \mathbf{p}^{(i+1)} - \mathbf{p}^{(i)}$ be a vector in the direction of the path. $\mathbf{p}^{(i)}$ moves in the positive gradient direction perpendicular to $\Delta\mathbf{p}^{(i)}$:

$$\mathbf{p}^{(i)} \leftarrow \mathbf{p}^{(i)} + \alpha\mathbf{q}^{(i)}, \quad \mathbf{q}^{(i)} = \mathbf{g}^{(i)} - (\mathbf{g}^{(i)} \cdot \Delta\mathbf{p}^{(i)}\|\Delta\mathbf{p}^{(i)}\|^{-1}) \tag{1}$$
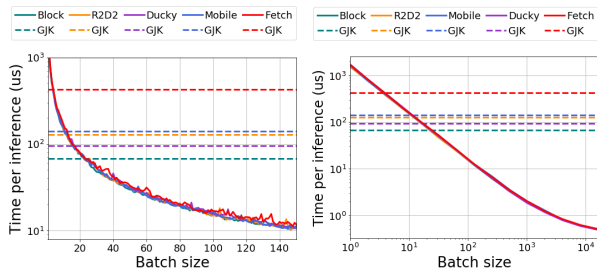

Algorithm 2 Repair

Input: $\mathbf{C}_{\hat{\theta}}$: Trained clearance estimator. α : step size. n_{ex} : extra steps. $t_{\max}$: Timeout.
Input / Output: $\mathcal{P}$: Sequence of poses from $(\mathbf{p}_{rs}, \mathbf{w})$ to $(\mathbf{p}_{rg}, \mathbf{w})$.
Output: Success: Boolean. Is the path certified valid?
 1: $i \leftarrow 0$, $n_{\text{shifts}} \leftarrow 0$
 2: **while** $0 < i <$ length$(\mathcal{P})$ and $n_{\text{shifts}} < t_{\max}$ **do**
 3:     **if** isValid$(\mathbf{p}^{(i)})$ **then** $i \leftarrow i + 1$; Continue.
 4:     **while** not isValid$(\mathbf{p}^{(i)})$ and $n_{\text{shifts}} < t_{\max}$ **do**
 5:         Shift $\mathbf{p}^{(i)}$ toward safety by α according to Eq. (1). $n_{\text{shifts}} \leftarrow n_{\text{shifts}} + 1$
 6:     **end while**
 7:     Shift $\mathbf{p}^{(i)}$ another n_{ex} steps toward safety.
 8:     Add unchecked interpolation points between new $\mathbf{p}^{(i)}$ and $\mathbf{p}^{(i-1)}$, $\mathbf{p}^{(i+1)}$.
 9:     $i \leftarrow i$ Now refers to new next point after $\mathbf{p}^{(i-1)}$.
10: **end while**
11: **if** $i ==$ length$(\mathcal{P})$ **return** True, $\mathcal{P}$ /* Whole path is certified valid. */
12: **for** $j = i - 1; j > 0; j \leftarrow j - 1$ /* Back out and repair with RRT.*/ **do**
13:     **if** Fastron repair step [7] on P(j, P(length)) is successful **then** return True, $\mathcal{P}$.
14: **end for**
15: **return** False, $\mathcal{P}$ /* RRT could not find a valid path */

---

where α is step size and $\mathbf{q}^{(i)}$ is the orthogonal projection of gradient $\mathbf{g}^{(i)}$ onto the hyperplane orthogonal to the path vector $\Delta\mathbf{p}^{(i)}$.

Algorithm 2 describes the repair procedure. First, we check all points in the path with classic geometry-based collision checking (Line 3). If all points are valid the algorithm returns the unmodified path (Line 11). Otherwise for each point $\mathbf{p}^{(i)}$ in collision, we incrementally shift it towards safety according to (1) until it is no longer in collision (Lines 4-6). Next, we interpolate between any shifted points and repeat the process for these new points (Lines 8-9). This is not guaranteed to be successful, so after a certain number of attempts, we revert to Fastron's repair step for the non-validated portions of the path. Fastron's repair step takes a path, assuming the start and end are valid, and returns a repaired path that is guaranteed to be collision free w.r.t. traditional geometry-based collision checking. First, it uses traditional collision checking to find invalid path segments between valid point pairs. Next, it solves a series of small queries around invalid regions using RRT. Finally, the valid path is concatenated and returned. Our modification of the Faston repair step includes path's start and end points in the RRT search. The intuition behind the Fastron repair step is that queries between closer configuration points are easier to solve. While this is often true in practice, in general the smaller queries may be infeasible. To be able to find a solution if one exists, we perform a series of repair steps backing out through the portions of the path that are already certified valid (Lines 12-14). Finally, if the repair, which includes RRT query from the original start and end poses, fails to find a solution, we consider a query infeasible (Line 15).

(a) Time v. batch to 150   (b) Time v. batch to 16K

Fig. 3. ClearanceNet inference time over batch size and model. Both y-axes and the second x-axis are logarithmic. Max speedups were 132, 261, 287, 193 and 845x respectively for Block, R2D2, Mobile, Ducky and Fetch.

Fig. 4. ClearanceNet accuracy over training set size and DoFs for Block and Fetch environments.

5   Results

Setup: We evaluate ClearanceNet and CN-RRT on five environments (Fig. 2). *Block* and *R2D2* contain two 7-DoF Kuka arms with fixed bases. Block contains a floating object (Fig. 2a), R2D2 a model of R2D2 (Fig. 2b). Planning DoFs and ClearanceNet input length are 14. *Ducky* contains a fixed-base 7-DoF Kuka arm and three objects (Fig. 2c), selected to have different geometric properties, from the uniform block to the duck with smooth curved surfaces. The workspace is parameterized with the object locations. Thus planning is 7-DoF but the extended configuration space dimensionality (and ClearanceNet input length) is 16; 7 for the robot pose and 3 for each movable obstacle in the scene. *Mobile* consists of three mobile manipulators: Kuka arms attached to moving bases with 50 cm sides (Fig. 2d). The collision checks between the three mobile manipulators are done as three pair-wise checks. Motion planning is 30-DoF but ClearanceNet input length is 17 for each pair-wise check; 7×2 for arm joints, one for base separation and two for base orientations. *Fetch* contains a Fetch robot, two sets of shelves and cubes in a 2m×2m square. The shelves can slide and the cubes can be at any position on the floor. Motion planning is thus 11-DoF and ClearanceNet input length is 17.

5.1   ClearanceNet Results

Training: We generate datasets of 10^6 training and 10^4 evaluation samples for each environment using Bullet Physics Simulation [6], which computes separation distances using the GJK distance algorithm [9]. Data collection takes 17-30 minutes per environment. The neural networks have layers with 1400 neurons.

Hyperparameters: We use Vizier [10] with Gaussian Process Bandits [20] to search for learning hyperparameters using a total population of 1000 models. A single network trains in one hour, while the hyperparameter search takes 24 hours on average. One set of optimized hyperparameters performs well across the environments and robots; thus we perform the tuning only once: dropout rate = 1%, learning rate = 1.7495×10^{-4}, batch size = 191. The overhead for
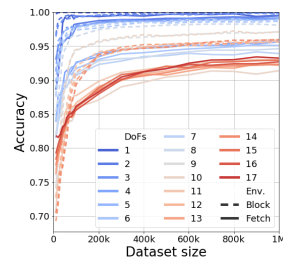
applying CN-RRT to new robots is less than two hours, for most of which the robot can continue to operate as the model trains offline.

Inference Speed: Time to perform a single collision check with ClearnanceNet decreases with batch size (Fig. 3), which we use to our advantage when building batched RRTs. It is interesting to note that, unlike classic methods, the network's size and therefore inference speed do not depend on the robot and workspace, suggesting that the method might be especially appropriate for environments with complex geometries and high-dimensional C-Spaces.

Accuracy: Fig. 4 shows ClearanceNet's accuracy for training set sizes up to 10^6, with various DoFs frozen, for the most complicated (Fetch) and least complicated (Block) environments. This shows the required number of samples depends on both degrees of freedom and workspace complexity. Fig. 5 displays ClearanceNet's accuracy with 10^6 samples and summarizes classifications for $d^* = 0$. False negatives can lead to trajectories with collisions being classified as free, requiring more repair time. False positives can lead to conservative planners that find no paths at all; the adaptive threshold addresses this.

5.2   CN-RRT Results

Setup: We compare CN-RRT to RRT [16] with GJK collision checking [9] (GJK-RRT), and RRT with Fastron [7] collision checking (Fastron-RRT). GJK is a geometry-based clearance finding algorithm, while Fastron is a learned collision checker that uses a support vector machine (kernel perceptron). Fastron-RRT includes a repair step that falls back on GJK-RRT to validate and correct paths.

ClearanceNet is implemented with TensorFlow using the Python interface. Bullet is implemented in C++, and we use the Python interface PyBullet. Fastron is implemented in Python using Numpy. All evaluations run on an Nvidia Tesla V100 GPU with 16 Gbs of RAM, with fixed start and goal configurations averaged over 100 problems. Thresholds and threshold switch times are determined per-environment using Vizier [10] with Gaussian Process Bandits [20].

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 (Fig. 6). 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 1).

Time allocation: In all trials, failure to solve a problem counts as the maximum time allotted. CN-RRT solves a single query faster than the baselines for all environments except Mobile (Fig. 6). Additionally, the repair time for CN-RRT is shorter than the build time for GJK-RRT on average, demonstrating that repairing a path is faster than building one from scratch.

Path length: Next, we examine the quality of the paths that CN-RRT finds. In all environments except Fetch, CN-RRT paths are shorter on average than GJK-RRT and Fastron-RRT (Table 1). Although we do not explicitly optimize for path length, ClearanceNet is biased toward higher-clearance paths, which are

Table 1. Path characteristics and path-building statistics for CN-RRT and baselines averaged over 500 queries. CN-RRT-NG is an ablated version of CN-RRT with no gradient repair. Succ. Rate is the method's success at finding a path within the time limit. Collision Checks are geometric checks, whereas Col. Checks Heuristic are checks by ClearanceNet and Fastron. The %Δ columns show percent increase or decrease as compared to GJK-RRT. The Path Length column group considers only successful queries, which biases it in favor of methods that solve only the shorter, easier problems.

Env.	Method	Succ. Rate (%)	Compute Time (s)			Path Length (steps)			Collision Checks		Col. Checks Heuristic	
			μ	%Δ	σ	μ	%Δ	σ	μ	σ	μ	σ
Block (Fig. 2a)	CN-RRT	**97.0**	**2.0**	-27	**5.2**	**964**	-21	**398**	**7.4k**	**41.9k**	**19.0k**	85.9k
	CN-RRT-NG	95.6	2.4	-14	6.3	1021	-16	413	14.8k	56.4k	20.7k	121.0k
	Fastron-RRT	52.0	15.7	+475	13.9	1169	-4	484	11.8k	48.7k	43.6k	**43.4k**
	GJK-RRT	95.2	2.7	-	6.7	1214	-	502	24.5k	67.8k	-	-
R2D2 (Fig. 2b)	CN-RRT	**98.8**	**1.6**	-42	**3.5**	**926**	-26	414	**0.5k**	**7.2k**	39.6k	122.2k
	CN-RRT-NG	98.2	2.0	-31	4.5	976	-22	442	5.4k	22.9k	43.1k	171.3k
	Fastron-RRT	97.6	3.4	+19	5.0	1170	-7	497	12.9k	34.3k	**3.8k**	**3.4k**
	GJK-RRT	97.6	2.8	-	5.2	1259	-	523	14.5k	31.0k	-	-
Ducky (Fig. 2c)	CN-RRT	**91.0**	**4.3**	-12	8.7	**673**	-36	336	18.8k	52.2k	**6.8k**	14.0k
	CN-RRT-NG	90.4	4.8	-2	9.0	941	-11	428	26.7k	59.2k	6.8k	11.3k
	Fastron-RRT	1.0	31.3	+545	**2.8**	741	-30	**309**	0.1k	**0.7k**	47.4k	**5.2k**
	GJK-RRT	89.4	4.9	-	9.2	1055	-	433	25.7k	51.3k	-	-
Mobile (Fig. 2d)	CN-RRT-NG	**70.7**	91.6	+2	67.3	**3799**	-10	**1824**	306.2k	265.4k	**295.4k**	524.6k
	Fastron-RRT	0.0	180.4	+101	**0.3**	-	-	-	**0.0k**	-	481.4k	**83.3k**
	GJK-RRT	63.5	**89.8**	-	73.4	4205	-	2099	411.1k	338.0k	-	-
Fetch (Fig. 2e)	CN-RRT-NG	**70.0**	**416.8**	-30	417.5	2857	+12	1454	436.7k	435.6k	**120.7k**	140.1k
	Fastron-RRT	8.0	923.1	+54	**266.7**	**1324**	-48	**781**	**24.8k**	163.8k	1268.2k	423.6k
	GJK-RRT	52.0	599.2	-	426.4	2553	-	1329	617.5k	439.3k	-	-

likely to be shorter in relatively uncluttered environments. In the more cluttered Fetch environment, GJK-RRT and Fastron-RRT have shorter paths on average, although this is affected by the fact that both algorithms solved only an easier subset of the problems CN-RRT solved.

Gradient-based repair: There are two parameters we can tune: step size α and extra steps n_{ex}. 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 $\alpha = 0.15$ and $n_{ex} = 0$. For R2D2 and Ducky, $\alpha = 0.05$ and $n_{ex} = 3$. Gradient-based repair is inapplicable to Mobile because the collision checks are done pair-wise, and to Fetch because the robot is non-holonomic (the base cannot translate sideways). For these environments we use the ablation method CN-RRT-NG. For all other environments, gradient-based repair is an improvement over GJK-based repair in terms of both average calculation time and average path duration (Fig. 6 CN-RRT-NG).

On robot experiments: We validate CN-RRT by taking paths it finds and executing them on a real Fetch robot. The start poses are randomly generated, and the goal poses are selected to be especially challenging: they require reaching into enclosed shelves (Fig. 1). For one randomly selected query, CN-RRT's planning time is 47 s, while the comparison methods fail to find a solution.

6 Analysis and Discussion

Probabilistic completeness: Algorithm 1 is probabilistically complete. We can consider three cases for the $C_{\hat{\theta}}$: 1) the classifier classifies all points as in col-

(a) Block, 95.5% Input size: 14

(a) Block, Success rate

(b) Block, Time allocation

(b) R2D2, 96.0% Input size: 14

(c) R2D2, Success rate

(d) R2D2, Time allocation

(c) Ducky, 91.4% Input size: 16

(e) Ducky, Success rate

(f) Ducky, Time allocation

(d) Mobile, 96.3% Input size: 17

(g) Mobile, Success rate

(h) Mobile, Time allocation

(e) Fetch, 95.1% Input size: 11

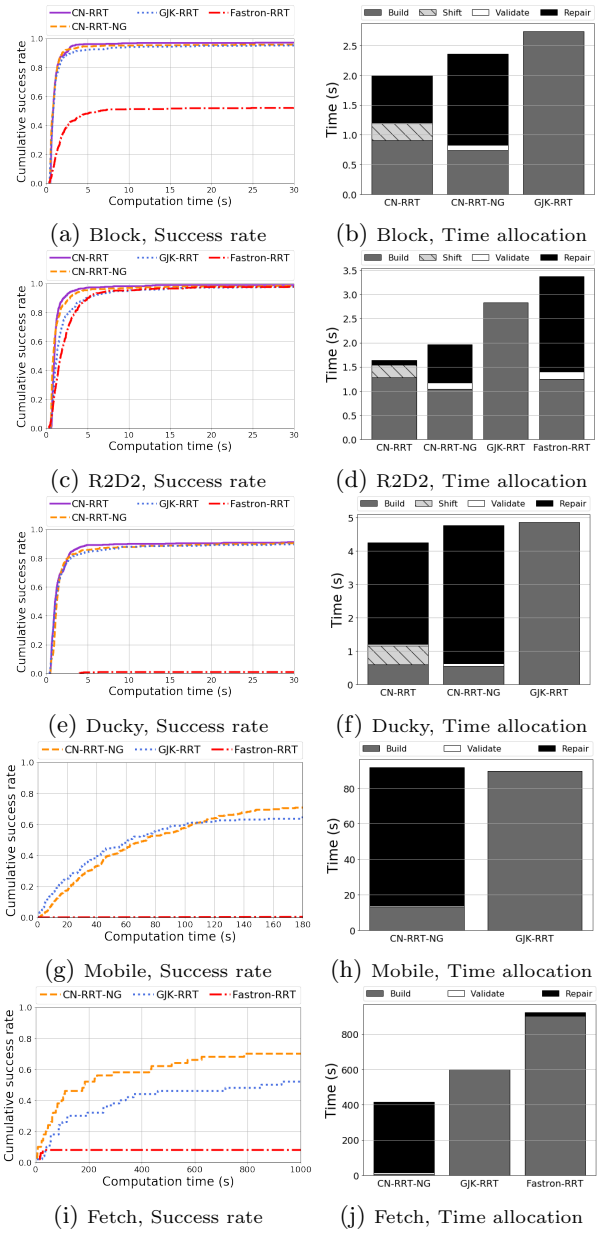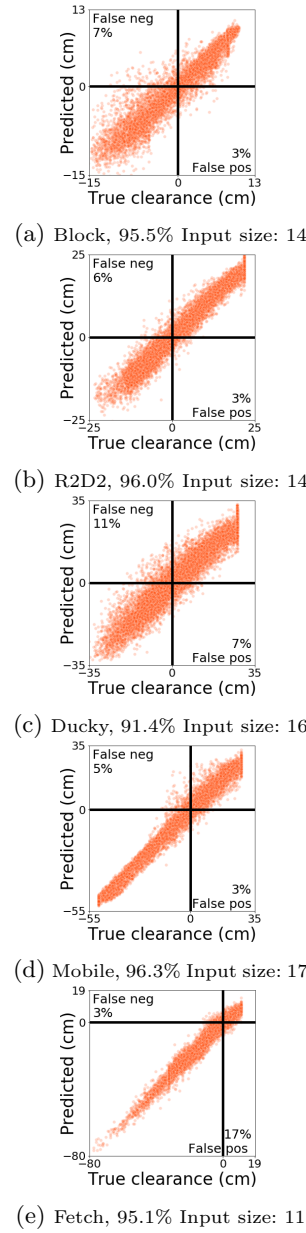(i) Fetch, Success rate

(j) Fetch, Time allocation

Fig. 5. ClearanceNet accuracy. Quadrants are true positive collision (bottom left), true negative collision-free (top right), false positive (bottom right), and false negative (top left).

Fig. 6. Success rate over time and time per phase. The bar graphs show average time spent in each phase: constructing the initial RRT (Build), repairing the path with the network gradient (Shift), checking the path with GJK (Validate) and replanning broken segments using GJK-RRT (Repair). Fastron is included when its scale is comparable to the other methods.

lision, 2) all points as out of collision, and 3) $C_{\hat{\theta}}$ falls somewhere in the middle. In the first case, CN-RRT will time out and proceed with its repair loop (Algorithm 2 Lines 12-14) acting from $\mathbf{p}_{rs}$ to $\mathbf{p}_{rg}$. In the second case, the algorithm reduces to a lazy RRT and again falls back on the repair loop. In the last case, Algorithm 1 cheaply finds a candidate path which is passed to Algorithm 2 for verification and possible repair. The invariant for the while loop (Algorithm 2 Lines 2-10) is that all points in path $\mathcal{P}$ with indices $[0, .., i-1]$ are certified valid with GJK. Upon exit, the algorithm either returns because the whole path is certified valid, or it enters the repair loop. The invariant of the repair loop is that $\mathcal{P}[0, j]$ is certified valid and $\mathcal{P}[j, \mathcal{P}.length]$ is invalid. Our modification of Fastron repair performs RRT with geometry-based collision checking on $\mathcal{P}[j, \mathcal{P}.length]$. Therefore if the repair step of Line 13 is successful, it returns path $\mathcal{P}[0, j]$ and $\mathcal{P}[j, \mathcal{P}.length]$ is valid. Otherwise, probabilistically complete RRT found no path between $\mathcal{P}[0, \mathcal{P}.length]$, and we return that the query was infeasible on Line 15 in Algorithm 2. Thus, Algorithm 1 is probabilistically complete and returns a valid path iff RRT with geometry-based collision checking finds a path.

Neural networks as heuristics: Separation distance is a Lipschitz continuous function when the robots and obstacles are rigid bodies [15]. CN-RRT exemplifies several benefits of choosing a neural network as a function approximation of a Lipschitz continuous function. First, neural networks running on GPUs are exceptionally efficient at processing large batches of data, and conveniently motion planning is well-suited for batched processing [1]. Second, neural networks are differentiable functions, and thus the decision making manifold (contact surface defined with threshold d^* selection) is "well-behaved" (Lipschitz continuous and monotonic), allowing a simple and effective algorithm conservativeness tuning. Last, the differentiability of the approximator comes in handy for recommending incremental path improvements in the direction of its gradient. Future work can look into approximating other complex-to-compute Lipschitz continuous measures in motion planning and exploit the same benefits (large data batching, decision-making manifold adaptation and gradient-based improvements) to make motion planning more efficient.

Run-time trade-off analysis: The biggest downside of neural networks as approximators is the one-time cost of data collection and training, which may be justified if they are used in enough queries. We explore the trade-offs between cost of training, collision checking speed up and number of collision checks, and answer how many queries are needed to offset the cost of training. Training and data collection can be, and in practice often are, done offline in a process concurrent to robot operations, without blocking a robot. Thus, the number of queries N needed to offset neural network overhead is $N > \frac{t_{\text{invest}}}{\Delta t} + \frac{t_{\text{offline}}}{t_{\text{rrt}}}$, $\Delta t = t_{\text{rrt}} - t_{\text{cnrrt}}$, where $t_{\text{invest}}, t_{\text{offline}}$ are online and offline time investment and $t_{\text{rrt}}, t_{\text{cnrrt}}$ are expected query completion time for RRT and CN-RRT.

The cost of motion planning with RRTs is $t_{\text{rrt}} \sim n_{\text{checks}} * t_{\text{check}}$, where n_{checks} = expected number of collision checks and t_{check} = time for a single geometric collision check. The time to solve one CN-RRT query is $t_{\text{cnrrt}} \sim n_{\text{checks}} * f_{\text{fp}} * f_{\text{batch}} * t_{\text{check}} + f_{\text{fn}} * t_{\text{check}} * f_{\text{pathlen}}$, where: $f_{\text{fp}} \geq 1$ is the in-

crease factor in collision checks due to $C_{\hat{\theta}}$ mistakenly classifying non-collisions as collisions (i.e. the cost of false positives); $f_{\mathrm{fn}} \geq 1$ is the increase in collision checks due to points being misclassified as free, triggering repair (i.e. the cost of false negatives); $f_{\mathrm{pathlen}} << 1$ is the ratio of expected path length to total number of collision checks; f_{batch} is computation speed up due to batching, $0 < f_{\mathrm{batch}} << 1$, as a ratio between time to compute a single collision check with a neural network (in a batch) and t_{check}. After substituting the above and arranging the terms, we arrive at:

$$N > \frac{t_{\mathrm{invest}}/t_{\mathrm{check}}}{n_{\mathrm{checks}}(1 - f_{\mathrm{fp}}f_{\mathrm{batch}} - f_{\mathrm{fn}}f_{\mathrm{pathlen}})} + \frac{t_{\mathrm{offline}}/t_{\mathrm{check}}}{n_{\mathrm{checks}}} \qquad (2)$$

The cost of training consists of time for data collection ($\|\mathcal{D}\|t_{\mathrm{check}}$) and time for training ($f_{\mathrm{train}}t_{\mathrm{check}}$), where $\|\mathcal{D}\|$ = dataset size and $f_{\mathrm{train}} > 1$ is the training time expressed as a multiple of t_{check}. We consider two options: a) data collection is online and training is offline ($t_{\mathrm{invest}} = \|\mathcal{D}\|t_{\mathrm{check}}$ is time for data collection, and $t_{\mathrm{offline}} = f_{\mathrm{train}}t_{\mathrm{check}}$ is time for training), and b) both are online ($t_{\mathrm{offline}} = 0$, $t_{\mathrm{invest}} = (\|\mathcal{D}\| + f_{\mathrm{train}})t_{\mathrm{check}}$). See Fig. 1 in the supplementary material for an illustration. Eq. (2) yields the following conclusions:

1. *Precision/recall trade off:* The classifier's false negative rate is more important, since $f_{\mathrm{pathlen}} >> f_{\mathrm{batch}}$, justifying the adaptive thresholding in Alg. 1. Here, f_{pathlen} in $10^{-2} - 10^{-1}$ range (Table 1), while $f_{\mathrm{batch}} \sim 10^{-3}$ (Fig. 3).
2. *Training trade-off:* There is a linear dependency between training time and data collection. Thus, it is worthwhile to look for more sample-efficient and faster training methods. In this paper, data collection requires about 30 minutes, and training about 60 minutes for a total of 90 minutes of overhead.
3. *Motion planing complexity:* CN-RRT fares better at complex queries that require many collision checks, because N is inversely proportional to n_{checks}.
4. *Knowing N* allows us to tailor the training. For small N, we can use cheap training and inaccurate models (increasing f_{fp} and f_{fn}); for large N, we can invest in longer, more precise training. This suggests future research in active sampling and interactive training. For our environments, we estimate $t_{\mathrm{rrt}}, t_{\mathrm{check}}, \Delta t, \|\mathcal{D}\|$ from Table 1 and Fig. 4 and derive that in the the fully online setting N is in the thousands, with the exceptions of Fetch, which requires fewer than 25 queries, and Mobile, which provides no savings. In the more realistic, offline training case, the required number of queries drops to 5 for Fetch, and less than a thousand for the simpler environments. See Supplemental materials for the full analysis.

Sampling vs. inference complexity: Interestingly, f_{batch} depends only on DoFs and not on the geometric properties of workspace objects. The topological complexities of the configuration space are addressed through the training set. For a good separation distance approximator, it is imperative that the training samples are from relevant and interesting regions. This suggests future work applying different sampling heuristics to collect high-quality training sets. Here we use uniform random sampling, but more appropriate sampling would likely result in better models, or require fewer samples to achieve the same accuracy.

Limitations: The two main features missing from CN-RRT are adaptation to moving obstacles and adaptation to new workspace objects. CN-RRT can easily be adapted for moving obstacles; the only change required is to let the workspace configuration $\mathbf{w}$ change during motion planning in Algorithm 1. The second is less straightforward. One avenue to address this is to use domain adaptation and learn to extend the network for additional degrees of freedom and perform additional training. Another approach is to use sensor input, which provides up-to-date information on the environment, but may also be high-dimensional.

Future work: An interesting extension would be to implement a lifelong planner that starts as a classic planner, then over its lifetime trains a collision checking model with data collected during planning, or generalizes between different robots of similar geometries. A different direction could make the gradient repair step in a specific direction to target specific robot joints.

7   Conclusions

We presented ClearanceNet, a neural network approximator for separation distance in extended configuration space, and CN-RRT, an RRT algorithm that exploits efficient batch processing and neural network differentiability. CN-RRT adds multiple edges at a time, adaptively relaxes clearance requirements for difficult queries, and uses ClearanceNet's gradient for repairs. Evaluated on five environments, CN-RRT produces shorter paths more quickly. On-robot experiments demonstrate its applicability to real robots. Finally, we analyzed the algorithm's trade-offs and offered ideas for future work using neural networks as approximators for Lipschitz continuous measures in motion planning.

8   Acknowledgements

References

1. 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.
2. 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.
3. 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.
4. L. Chiang, A. Faust, S. Sugaya, and L. Tapia. Fast swept volume estimation with deep learning. *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2018.
5. 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.

6. E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2019.
7. N. Das and M. Yip. Learning-based proxy collision detection for robot motion planning applications. *IEEE Transactions on Robotics*, pages 1–19, 2020.
8. M. Elbanhawi and M. Simic. Sampling-based robot motion planning: A review. *IEEE Access*, 2:56–77, 2014.
9. 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.
10. 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.
11. K. E. Hoff, A. Zaferakis, M. Lin, and D. Manocha. Fast and simple 2d geometric proximity queries using graphics hardware. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, page 145–148, New York, NY, USA, 2001. Association for Computing Machinery.
12. 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.
13. 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.
14. S. Kumar, S. Choudhary, and S. Srinivasa. Estimating configuration space belief from collision checks for motion planning. *CoRR*, abs/1901.07646, 2019.
15. S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.
16. S. M. LaValle and J. James J. Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.
17. M. C. Lin and J. F. Canny. A fast algorithm for incremental distance calculation. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1008–1014, 1991.
18. 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.
19. J. Pan, X. Zhang, and D. Manocha. Efficient penetration depth approximation using active learning. *ACM Transactions on Graphics*, 2013.
20. 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.
21. T. N. Tran, J. Denny, and C. Ekenna. Predicting collision detection with a two-layer neural network. 2019.
22. F. Xia, W. B. Shen, C. Li, P. Kasimbeg, M. Tchapmi, A. Toshev, R. Martín-Martín, and S. Savarese. Interactive gibson: A benchmark for interactive navigation in cluttered environments, 2020.
23. M. Yavari, K. Gupta, and M. Mehrandezh. Lazy steering rrt*: An optimal constrained kinodynamic neural network based planner with no in-exploration steering. In *2019 19th International Conference on Advanced Robotics (ICAR)*, pages 400–407, 2019.