

426 A Implementation Details in training

427 From given training dataset \mathcal{D} , the training objective is $\sum_{d \in \mathcal{D}} L_{SVCD}(d)$, which is the binary cross-
 428 entropy loss between predicted collision outcomes and ground-truth labels y_{SVCD} . However, binary
 429 classification loss alone results in almost zero gradients when two objects are deeply penetrated
 430 into each other because the probability is close to 1 and nearly unchanged. This is undesirable
 431 because The training objective is defined as $\sum_{d \in \mathcal{D}} [L_{SVCD}(d)]$, where L_{SVCD} is the binary cross-
 432 entropy loss between predicted collision outcomes and ground-truth labels y_{SVCD} . , and L_{reg} is
 433 a regularization loss given by: $L_{reg}(d) = \left(\|\nabla f_{SVCD}(d)\| - 1 \right)^2$, encouraging smooth gradient
 434 variations beneficial for stable trajectory optimization [28].

435 B Implementation Details in SVCD Accuracy

436 **Problem Generation.** First, we compute the shortest distance vector δ between the moving robot
 437 and the fixed object by checking the penetration depth between the robot in a large number of
 438 intermediate configurations and the fixed object. The obstacle is then translated by $\delta + \epsilon$, where
 439 $\epsilon \sim \mathcal{N}(0, 0.03^2)$ represents a small perturbation. After translating the fixed object, we calculate
 440 the penetration depth as we did previously to check the collision label. During problem generation,
 441 rejection sampling is used to balance positive and negative collision labels.

442 **Sphere Approximation.** To approximate the shape of a moving object with a set of spheres, we
 443 follow strategies from cuRobo. A set of spheres is constructed by combining spheres placed inside
 444 the object from voxelized interior regions with spheres centered at uniformly sampled surface points.
 445 For this process, we can vary the size of the voxel and the number of sampled surface points as
 446 hyperparameters.

SVCD Method	Hyperparameter	Sampling Range
Convex Cell / GJK	Trajectory Discretization Number	$U(2, 256) \subset \mathbb{Z}$
	Activation Length	$U(-0.01, 0.01) \subset \mathbb{R}$
Sphere / Sphere-Mesh Distance	Trajectory Discretization Number	$U(2, 64) \subset \mathbb{R}$
	Activation Length	$U(-0.01, 0.01) \subset \mathbb{R}$
	Voxel Size (Sphere Approximation)	$U(0.01, 1) \subset \mathbb{R}$
	Number of Surface Points (Sphere Approximation)	$U(0, 100) \subset \mathbb{Z}$
	Mesh Simplification Voxel Factor	$U(2, 32) \subset \mathbb{Z}$
NeuralSVCD	Trajectory Discretization Number	$U(2, 100) \subset \mathbb{Z}$
	Reduce K	$U(2, 256) \subset \mathbb{Z}$

Table 2: Hyperparameter Sampling Ranges for SVCD Baselines. $U(\min, \max)$ denotes uniform distribution

Algorithm 1 Proposed SVCD Algorithm

Require: Static Mesh: mesh_{static} , Moving Mesh: mesh_{mov} , Trajectory: τ

Ensure: Collision prediction output: ccd_output

```

1:  $\mathcal{Z}_{static}, \mathcal{Z}_{mov} \leftarrow \text{encode}(\text{mesh}_{static}, \text{mesh}_{mov})$ 
2:  $\{(p_1^{(i)}, r_1^{(i)}, z_1^{(i)}, p_2^{(i)}, r_2^{(i)}, z_2^{(i)}), t^{\dagger(i)}\}_{i=1}^M \leftarrow \text{broadPhase}(\mathcal{Z}_{static}, \mathcal{Z}_{mov}, \tau)$ 
3:  $\text{SVCD\_output} \leftarrow -\infty$ 
4: for  $i = 1$  to  $M$  do
5:    $\xi^{(i)}, \tau(t^{\dagger(i)}) \leftarrow \text{LinearApproximation}(\tau, t^{\dagger(i)})$ 
6:    $\text{SVCD\_logit}^{(i)} \leftarrow f_{SVCD}(p_1^{(i)}, p_2^{(i)}, z_1^{(i)}, z_2^{(i)}, \xi^{(i)}, \tau(t^{\dagger(i)}))$ 
7:    $\text{SVCD\_output} \leftarrow \max(\text{SVCD\_output}, \text{SVCD\_logit}^{(i)})$ 
8: end for
   return  $\text{SVCD\_output}$ 

```

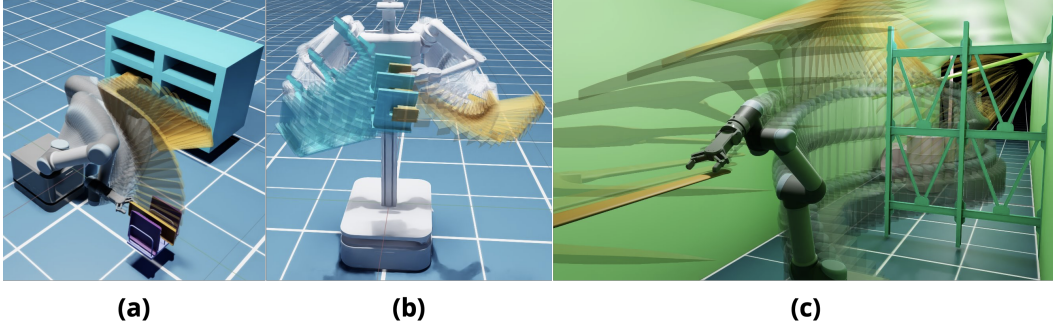


Figure 7: Illustration of three robotic tasks solved by our proposed algorithm. (a) Dish insertion: A UR5 robot precisely inserts dishes into a dish rack. (b) Peg assembly: ARMADA [25], a 12-DOF bimanual manipulator, simultaneously holds a peg in one arm and a slot in the other, accurately assembling the peg into the designated slot. (c) Mobile manipulation at a mine tunnel: A mobile manipulator transports a pickaxe to their target location, carefully navigating around obstacles such as beams and a wagon. Each task highlights the critical importance of precise, collision-free motion planning.

447 C Domain Description for Motion Planning

448 **Dish insertion:** A fixed-base UR5 must insert three distinct dishes (sampled without replacement
449 from six shapes) into a rack. We randomly place the rack in a collision-free pose and sample a
450 collision-free robot start configuration. Hard-coded target poses specify both the end effector and
451 the final location of each dish. After each insertion, physics is enabled to let the dish settle; once
452 settled, it becomes a static obstacle for subsequent insertions.

453 **Bimanual insertion:** The ARMADA system, with two 6 DoF arms, performs a peg-into-slot as-
454 sembly. We generate collision-free start and goal configurations via rejection sampling: for each
455 trial, we sample a peg pose, grasp pose, and peg-hole shape, solve the inverse kinematics for joint
456 angles, and discard any samples that collide.

457 **Mining site navigation:** A UR5 on a mobile base carries a pickaxe through a synthetic mining
458 tunnel, avoiding wagons and beams. In each episode, we randomly select four meshes from eight
459 obstacle candidates and place them at fixed scene locations. The robot’s start configuration is sam-
460 pled from a bounded, collision-free region, while the pickaxe’s goal location remains constant.

461 D Implementation Details in Motion Planning

462 To demonstrate improvements in motion planning performance, we incorporate *NeuralSVCD* into
463 a trajectory optimization framework modeled after cuRobo [6]. In this framework, trajectories are
464 parameterized using splines with a fixed number of control points. Our method adopts a two-stage
465 optimization process. First, a particle-based solver (MPPI [29]) is employed to promote exploration,
466 and then the trajectory is refined using L-BFGS [30].

467 Initially, we represent the trajectory as a spline with a fixed number of control points $\tau_{control} \in$
468 $\mathbb{R}^{M \times F}$, where M is a number of control points, and F is a number of actuation of the robot. We
469 then apply an optimization-based motion planner with an objective function from cuRobo but with
470 custom collision cost:

$$C_{traj}(\tau_{control}) = C_{smooth}(\tau_{control}) + \alpha_{col} C_{SVCD}(\tau_{control}) \quad (2)$$

471 where C_{SVCD} and α_{col} denotes SVCD cost computed using *NeuralSVCD* and its coefficient. For
472 $C_{smooth} = \alpha_{vel} C_{vel} + \alpha_{acc} C_{acc} + \alpha_{jerk} C_{jerk}$, where each term indicates velocity, acceleration,
473 and jerk minimization cost with pre-defined coefficient. For cuRobo-*NeuralSVCD* and cuRobo-
474 *NeuralSVCD-discrete*, we modify the collision cost term in the loss function (2) by replacing it with
475 collision logits computed by f_{cd} . For each task, we use different hyperparameters. See Table 3, 4,
476 5. We also modify the calculating gradient during L-BFGS to be the bundled gradient [31], which

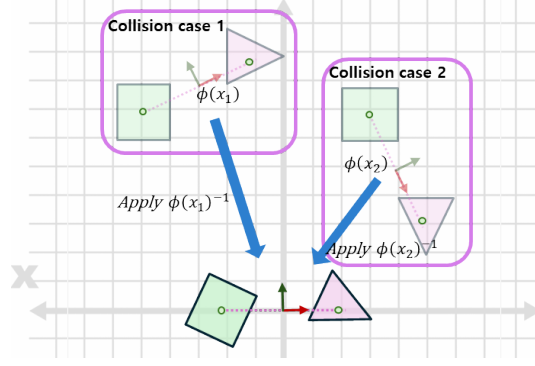


Figure 8: Illustration of pre-processing for achieving invariance. Case 1 and 2 have the same relative transform between two objects, but their global transforms are different. If we treat the two objects as a single composite rigid body, we can assign frames $\{\phi(x_1)\}$ and $\{\phi(x_2)\}$ whose origin is at the mid-point of the centers of two objects, and whose direction is determined by the line intersecting the centers. We then apply $\phi(x_1)^{-1}$ and $\phi(x_2)^{-1}$ to these frames so that they are at the origin of the world frame, with their orientation aligned with that of world frame as shown in the bottom. This preprocessing step ensures consistent input irrespective of the objects’ global poses.

combines gradient evaluations from multiple samples or time steps into a single aggregated update to improve optimization efficiency and stability, benefiting all baselines.

E Applying transformation to the latent vectors

Another critical consideration in designing latent representations and the neural SVCD decoder is the application of rigid-body transformations. During collision evaluation, each robot link must be positioned according to specific trajectory configurations, requiring transformations described by $SE(3)$ —a combination of rotations and translations. Using explicit mesh representations, applying transformations is straightforward: one simply applies transformations directly to mesh vertices before checking for collisions. However, when using latent representations z for robot links, applying $SE(3)$ transformations is not trivial, as latent representations do not inherently support direct geometric transformations.

One option is to use neural network-based transformation operators [32], which receive both the transformation and z and output the transformed z . However, they do not guarantee the rigid-body transformation properties required. For instance, applying a 180° rotation about the z -axis twice should reproduce the original representation, a consistency that such operators cannot reliably ensure.

To overcome these limitations, we introduce an analytical transformation operation within a high-dimensional latent space that explicitly preserves rigid transformation properties. Applying a rigid-body transformation $T \in SE(3)$ analytically involves separately treating translation and rotation. Given $T = (R, t)$, with $R \in SO(3)$ being a rotation matrix and $t \in \mathbb{R}^3$ a translation vector, we define the transformation of each point-latent pair as: $T \cdot (p_i, z_i) := (R \cdot p_i + t, D(R) \cdot z_i)$ where $D(R) \in \mathbb{R}^{K \times K}$ is a rotation operator applied directly within the latent space, explicitly constructed following the equivariant rotation formulation proposed in [33]. This operator $D(R)$ ensures that latent representations transform consistently with geometric rotations, preserving rigid-body properties. Intuitively, this means translations shift only the spatial coordinates p_i , while rotations alter the latent encodings z_i , effectively rotating the local geometric feature descriptions encoded within each latent vector.

F Implementation of encoder and neural SVCD decoder

For any two meshes and their corresponding moving direction ξ , the SVCD result is invariant under both $SE(3)$ transformations and scaling. In other words, if a consistent transformation T is applied

to the meshes and the moving direction or if a uniform scale s is applied, the collision detection outcome remains unchanged. We leverage this property when designing our neural CCD decoder. The inputs to the neural CCD decoder, f_{SVCD} , include the latent representations z_i , positions p_i , transformations T_i , and twists ξ_i (which contain both linear and angular velocities) at critical collision times determined during the broad-phase.

Building on ideas from [33, 34, 35], we first preprocess these inputs to achieve invariance with respect to additional transformations and scaling. The preprocessing pipeline is illustrated in Figure 8. Let the combined input be $x = [\xi; z_1; z_2; p_1; p_2]$. We define the preprocessing function as

$$\Phi(x) = \frac{1}{\max(\|z_1\|, \|z_2\|)} (\phi(x)^{-1} \cdot x),$$

with $\phi : \mathbb{R}^M \rightarrow SE(3)$ such that $\phi(T \cdot x) = T \cdot \phi(x) \quad \forall T \in SE(3)$, where M is the dimension of x and $\|z\|$ is Euclidean norm of z . It is straightforward to verify that Φ is invariant under any transformation in $SE(3)$ and scale. We define ϕ analytically by setting its translation component to $-(p_1 + p_2)$ and aligning its rotation $R \in SO(3)$ with the line connecting p_1 and p_2 , thereby rigidly attaching the transformation to the pair of bodies. This construction ensures the property $\phi(T \cdot x) = T \cdot \phi(x)$ holds.

The preprocessed inputs are then passed through multiple multilayer perceptrons (MLPs). The network outputs a scalar logit that represents a binary collision prediction (with positive logits indicating a collision). Finally, after predicting collisions for every p_i and p_j pair identified during the broad-phase, we apply max pooling to obtain the final logits indicating a collision between the rigid bodies.

The shape encoder processes an input point cloud to produce N local representations $\{(z_i, p_i)\}_{i=1}^N$. The sample positions $\{p_i\}$ are selected via Furthest Point Sampling (FPS), and the corresponding latent vectors $\{z_i\}$ are learned by a neural network. Concretely, every surface point is assigned to its nearest sample p_i , and all points in each partition are fed—independently but using shared weights—through the encoder to generate the local z_i . To ensure compatibility with our latent-space transformations, we adopt the FER-VN-OccNet encoder architecture from [33].

Hyperparameter	cuRobo-sphere-100	cuRobo-sphere-1200	cuRobo-NeuralSVCD	cuRobo-NeuralSVCD-discrete
Number of initial seeds			2	
MPPI - interpolation number		4		100
MPPI - iteration			40	
MPPI - number of samples		200		50
MPPI - number of control points			5	
LBFGS - interpolation number		2		100
LBFGS - iteration			10	
LBFGS - number of control points			10	
α_{vel}			10.0	
α_{col}			1.0	
α_{acc}			2	
α_{jerk}			10	
Activation distance		0.010	-	-
Number of sphere for robot		891	-	-
Number of sphere for grasping object	100	1200	-	-

Table 3: Hyperparameter settings for the bimanual insertion task across motion planners.

Hyperparameter	cuRobo- sphere- 50	cuRobo- sphere- 2000	cuRobo- <i>NeuralSVCD</i>	cuRobo- <i>NeuralSVCD</i> - discrete
Number of initial seeds			2	
MPPI - interpolation number		3		100
MPPI - iteration			40	
MPPI - number of samples		400		50
MPPI - number of control points			8	
LBFGS - interpolation number		8		100
LBFGS - iteration			10	
LBFGS - number of control points			3	
α_{vel}			1	
α_{col}			5	
α_{acc}			10	
α_{jerk}			50	
Activation distance	0.040	0.040	-	-
Number of sphere for robot		397	-	-
Number of sphere for grasping object	50	2000	-	-

Table 4: Hyperparameter settings for the mining site navigation task across motion planners.

Hyperparameter	curobo- sphere- 50	curobo- sphere- 400	curobo- <i>NeuralSVCD</i>	curobo- <i>NeuralSVCD</i> - discrete
Number of initial seeds			2	
MPPI - interpolation number		4		50
MPPI - iteration			40	
MPPI - number of samples		200		50
MPPI - number of control points			5	
LBFGS - interpolation number		2		50
LBFGS - iteration			10	
LBFGS - number of control points			10	
α_{vel}			10.0	
α_{col}			1.0	
α_{acc}			2	
α_{jerk}			10	
Activation distance	0.040	0.040	-	-
Number of sphere for robot		397	-	-
Number of spheres for grasping object	50	400	-	-

Table 5: Hyperparameter settings for the dish insertion task across motion planners.