Contact Localization for Robot Arms in Motion without Torque Sensing

Jacky Liang[1], Oliver Kroemer[1]

***Abstract*— Detecting and localizing contacts is essential for robot manipulators to perform contact-rich tasks in unstructured environments. While robot skins can localize contacts on the surface of robot arms, these sensors are not yet robust or easily accessible. As such, prior works have explored using proprioceptive observations, such as joint velocities and torques, to perform contact localization. Many past approaches assume the robot is static during contact incident, a single contact is made at a time, or having access to accurate dynamics models and joint torque sensing. In this work, we relax these assumptions and propose using Domain Randomization to train a neural network to localize contacts of robot arms in motion without joint torque observations. Our method uses a novel cylindrical projection encoding of the robot arm surface, which allows the network to use convolution layers to process input features and transposed convolution layers to predict contacts. The trained network achieves a contact detection accuracy of 91.5% and a mean contact localization error of 3.0cm. We further demonstrate an application of the contact localization model in an obstacle mapping task, evaluated in both simulation and the real world.**

I. INTRODUCTION

For robot manipulators to robustly operate in unstructured environments, safely interact with humans, and perform contact-rich tasks, they must be able to sense the contacts they make with the external environment. Indeed, works in tactile sensing have seen many uses in robot manipulation [1], including object localization [2], [3], [4], shape completion [5], [6], [7], and active environment exploration [8], [9], [10]. For robot skins, recent developments demonstrate promising applications in contact estimation [11], [12], whole-body manipulation [13], compliant control [14], [15], safe human-robot interactions [16], [17], object classification [18], [19], [20], and manipulation in dense clutter [21].

However, while robot skins is an area under active research and development [22], robust and affordable skins that work across a wide variety of robot form factors remain inaccessible. By contrast, proprioceptive sensing that gives joint angles and velocities, and sometimes estimated torques, are available in most commercial manipulator arms. As such, prior works have explored using proprioceptive sensors to localize contacts, with proposed methods ranging from model-based optimization [23], [24], [25], [26] to model-free learning [27], [28], [29].

Many past works take the perspective of localizing contacts on a static robot arm - the arm is initially set still, then one or more point contacts are established, pushing in the direction normal to the robot mesh. A typical additional assumption is that the point contact stays in the same position on the robot arm, even if the arm moves as a result of the contact. While this approach is useful in the context of "passive" contact incidence, where the the obstacles "come" to the robot, we argue it is less realistic in the context of "active" contact incidence, where the robot "goes" to the obstacle, whether for exploration or manipulation.

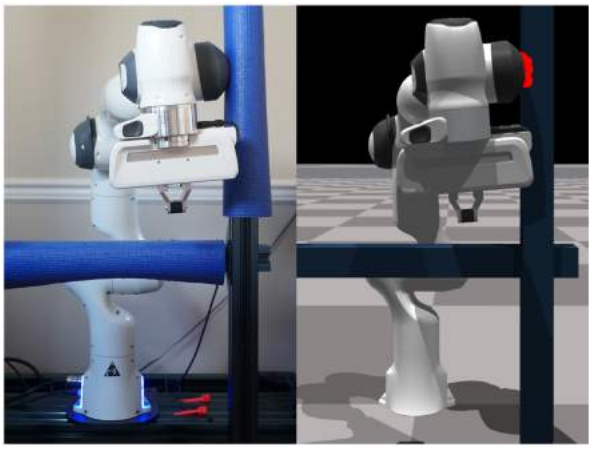

Fig. 1: A learned neural network contact localization model predicts contact points, visualized in red, on the Franka robot. The model is trained in simulation via Domain Randomization, and it does not require torque observations. To aid sample complexity and localization performance, the network uses transposed convolution layers to predict contact distance fields in the cylindrical projection space of the mesh surface.

In this paper, we train a neural network to localize contacts on the surface of a robot arm that is in motion and interacting with obstacles. The neural network is trained with data collected in simulation, where ground truth contact information can be obtained. When a robot is in collision with an external obstacle, its proprioceptive responses vary depending on the dynamics model and inertial parameters of the robot. Because there are mismatches in simulated and real dynamics (sim-to-real gap), we use Domain Randomization [30] to generate data across a distribution of robot dynamics models.

Furthermore, because of the sim-to-real gap, our network does not rely on torque observations to perform contact localization. This relaxes the requirement of having accurate force-torque sensing, and it follows from the observation made in [26], that contact points with static obstacles must have zero velocities in the direction of the surface normal. While the algorithm in [26] solves for potential contact locations with joint velocities, our approach uses velocity observations as features to a neural network.

In addition, we use a novel representation to encode both features and contact localization predictions. This representation projects points on the surface of robot links to cylindrical

[1]Carnegie Mellon University. {jackyliang, okroemer}@cmu.edu

coordinates, allowing us to represent features and contact points as images. With this encoding, the model can use convolution layers to efficiently process features of points on link surfaces, such as point-wise link velocities. We also propose using transposed convolution layers to predict images of contact points, represented as distance fields, on the surface of a robot link. This is unlike previous works [27], [28], [29], which directly classified the contact state of each of these points. Both transposed convolution and classification-based variants are evaluated, and we demonstrate an application of the learned contact localization networks in an obstacle-mapping experiment, conducted in both simulation and the real world. See video and supplementary materials at `https://sites.google.com/view/ct-loc`

II. RELATED WORKS

Prior works have studied combining a robot's proprioceptive observations and dynamics model to infer the robot's contact state. The authors of [23] proposed the Contact Particle Filter (CPF), a model-based optimization method that filters for external contacts on a humanoid robot by observing its joint torque residuals - the difference between measured and expected joint torques. The algorithm approximates solving contact locations as a quadratic program (QP). For the CPF's observation model, the contact likelihood is proportional to the error of the QP's solution. For its dynamics model, it assumes that contacts occur at fixed points on and relative to the robot surface - this means the contact point follows the robot link's movements. While the CPF is efficient and achieves good error rate (0.4s per filter step for 3 concurrent contacts, 2cm localization error), it assumes access to an accurate dynamics model of the robot.

Later works approached contact localization via torque residual observations from a data-driven perspective. In [31] the authors devise a real-time collision detection and localization algorithm for a humanoid robot by training a Support Vector Machine (SVM). A one-class SVM was used for detection, and multi-class SVM for localization. The work assumes one contact at a time, and the SVMs were trained on real-world data. While the detection model was trained with data where the robot arm is moving, the localization model assumed static robot configuration - the robot is not moving when the contact occurs. Furthermore, the localization model predicts only coarse labels - 7 classes for the entire arm of the humanoid. Additional contact detection works that focused on locomotion robots improved performance of contact detection, but not localization [32], [33].

In [27] the authors improved the resolution of data-driven contact localization by training machine learning models (Random Forests and Multi-layer Perceptrons) to classify the contact state of 661 pre-specified points on a 7-DoF Jaco arm's surface. The work assumes single contacts, that contacts are perpendicular to the robot's surface normals, and that there are no contact torques. The features of the model consist of a sliding window of joint positions, velocities, accelerations, torques, and linear accelerations, and data was generated in simulation. The best-performing model achieved a mean localization error of 4cm with 14% False Negative Rate (FNR) (not detecting a contact when there is one), and its inference frequency is higher than 200Hz, which is much faster than the optimization-based baseline. Like [31], the models also make the static configuration assumption, where the training data consists of 20 static arm configurations. The authors' follow-up work [25] proposed a particle filter approach that constrained particles of contact locations to the surface of the robot arm. This method achieved comparable performance to those in the earlier work, but it had the added capability of handling multiple contacts.

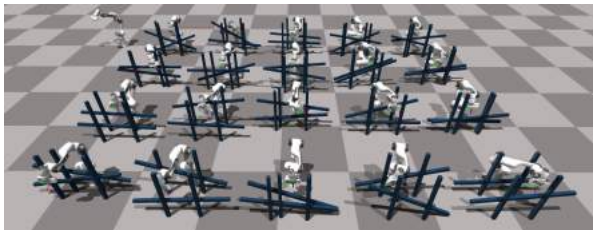

Fig. 2: Visualization of randomly sampled obstacle environments used for generating contact interaction data.

In [28] the authors propose a similar a data-driven approach to classify contacts on pre-specified points on a robot arm surface. Training data were generated in simulation, and like [27], this work also assumes one contact at a time, static robot configuration, and access to joint torque observations. The follow-up work [29] improved model performance by training with both simulation and real-world data, achieving 6.4cm mean localization error. In addition to detecting contacts, the algorithm in [34] can also classify between expected and unexpected contacts. However, the algorithm only gave coarse localization labels - one for the upper arm and one for the lower arm.

Recent works have also explored estimating external contacts without access to joint torque sensors. The authors of [24] propose a model-based algorithm that only needs end-effector force sensors and IMUs placed on the robot. The work also assumes a single contact, but the humanoid robot is moving during contact incident. In [26], the authors leverage joint velocity measurements to localize contacts. While the method achieves sub-centimeter localization error, it also assumes single contacts and was applied to a robot with planar kinematics - the links only moved in 2D. Our work is related to [24], [26] in that we do not assume access to joint torque readings. Like [27], [28], our approach predicts dense contact locations on the surface of 7-DoF robot arms, but our method also allows for predicting multiple contacts and predicting contacts while the robot is in motion.

III. METHOD

We define the contact localization problem as follows: given a sequence of T observations $o_{1:T}$ sampled at a frequency f, detect whether or not each link $l \in [1, \ldots, L]$ of the robot arm is in contact with an obstacle at time T, and if it is, also predict the contact locations $c_i^l \in \mathcal{S}_l$. Here, c_i^l denotes the ith predicted contact point of link l, which lies on the link's 3D surface manifold $\mathcal{S}_l$, and L is the total number

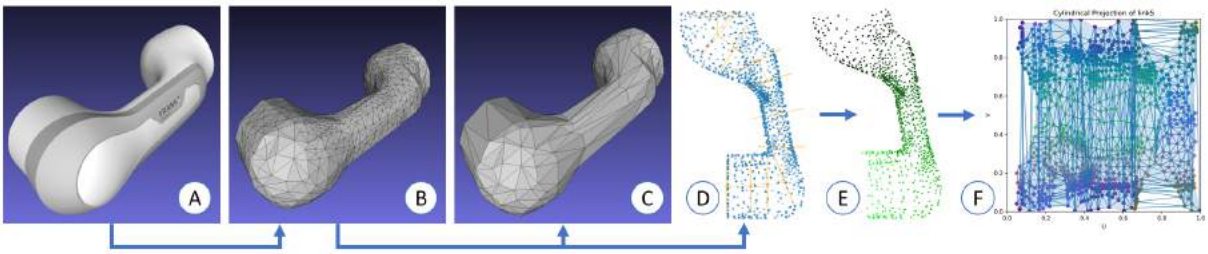

Fig. 3: Mesh simplification and cylindrical projection pipeline for link 5 on the Panda robot. A: Visual mesh provided by Franka Emika. B: Projection mesh with lower triangle count, more uniform vertex, and more equilateral triangles obtained via TetWild. C: Collision mesh obtained by decimating the projection mesh. D: Manually specifying waypoints for a tube (yellow lines and circles) to fit around the projection mesh (blue vertices). E: Projecting vertices onto the tube. Here the colors visualize the coordinates along the tube. Greener means the vertex is toward one end of the tube, and black the other. F: Normalized cylindrical projection coordinates in the range of $[0, 1]$. The horizontal U-axis denotes the coordinate along the tube, and vertical V-axis denotes the angular coordinate along the tube's circumference. Each dot is a vertex, and the color of the dot indicates its normal direction. Edges are mesh triangle edges. Blue shaded region denotes the concave hull used to define valid projection interpolation region.

of robot links for which contact localization is performed. We use $b^l \in \{0, 1\}$ to denote the binary contact state of each link. We assume the obstacles with which the robot makes contact are stiff, rigid, and stationary.

A neural network is trained to make this prediction. Training data is collected in simulation with a distribution of robot dynamics models. There are three important differences between our approach and previous works which also used machine learning to localize contacts for robot arms. First, our data is generated by having the robot arm interact with obstacles, instead of directly applying contact forces on the arm. This is important, because it leads to more realistic sequences of robot movements under contact - contact points often slide along the robot arm as it pushes against an obstacle. Second, we do not use torque observations and do not assume access to precise robot dynamics models, which makes our method more general and applicable to robots without accurate force/torque sensing. Third, we propose a novel cylindrical projection mapping to represent the robot link surface manifold, which can be used to encode both features and contact localization outputs.

A. Data Generation

We generate the training data by having a robot arm execute exploration trajectories in environments with randomly generated obstacles. The robot we use is the 7-DoF Franka Emika Panda arms. We collect features from and make predictions for the last 7 links on the Franka arm, excluding the first two base links, as they seldom come into contact with external obstacles. Obstacles in the scene consists of two sets of 3 beams, 2 vertical and 1 across, which are placed in front of the robot. We randomize the pose of beams, with the across beams attached to the vertical beams. This results in a diverse set of training environments (Figure 2).

There are two types of exploration trajectories the robot executes. The first we call random exploration, in which the robot randomly samples a sequence of delta end-effector pose targets and follows through each waypoint with randomly sampled time horizons. The second we call informed exploration, in which we predefine a sequence of waypoints that are likely to bring the robot in contact with obstacles in the scene, and the robot follows through each one with small amounts of added noise. Our training dataset contains data from both the random and informed exploration policies, with a $10 : 1$ ratio respectively. Details about trajectory generation can be found in the Appendix.

To go to each waypoint, the robot uses min-jerk interpolation and end-effector Cartesian-space impedance control, which converts errors in Cartesian space to torque commands via a spring-damper system. The simulation uses the Franka dynamics model from [35], which was fitted on a real Franka robot; this allows realistic impedance control behavior in simulation. We randomize the inertial parameters of the Franka dynamics model (mass, center of mass, and moment of inertia for each robot link) as well as the gains of the impedance controller for each trajectory.

We collect the following observations: joint angle velocities $\dot{q}_t \in \mathbb{R}^7$, linear velocities of each link $v_t^l \in \mathbb{R}^3$, angular velocities of each link $\omega_t^l \in \mathbb{R}^3$, difference between the current and target joint angles $\delta q_t \in \mathbb{R}^7$, difference between the current and target link poses $\delta p_t^l \in SE(3)$, difference between the current and "collision-free" joint angles and joint angle velocities $\Delta q \in \mathbb{R}^7, \Delta \dot{q}_t \in \mathbb{R}^7$, and the difference between the current and "collision-free" link poses and velocities $\Delta p_t^l \in SE(3), \Delta v_t^l \in \mathbb{R}^3, \Delta \omega_t \in \mathbb{R}^3$. We assume the robot can directly measure its joint angles and joint velocities, from which link velocities can be computed. The "collision-free" observations are obtained by running the simulation from the previous step in a separate simulation that has no obstacles. If the robot is not near an obstacle, then Δp_t, Δv_t, and $\Delta \omega_t$ are all zero vectors. Otherwise, some of these values might be non-zero, and they indicate how much the observed trajectory deviates from the expected trajectory if no obstacles were present. The target link poses are obtained by running forward kinematics on the desired joints $q_d = q + J^\top p_e$, where $q \in \mathbb{R}^7$ are the current joint angles, J is the analytical Jacobian, and $p_e \in SE(3)$ is the end-effector pose error used by the impedance controller. We do not use positional observations, like joint angles or end-effector poses. Avoiding them for training the network ensures that it is not overfitting to the locations of obstacles from the training data.

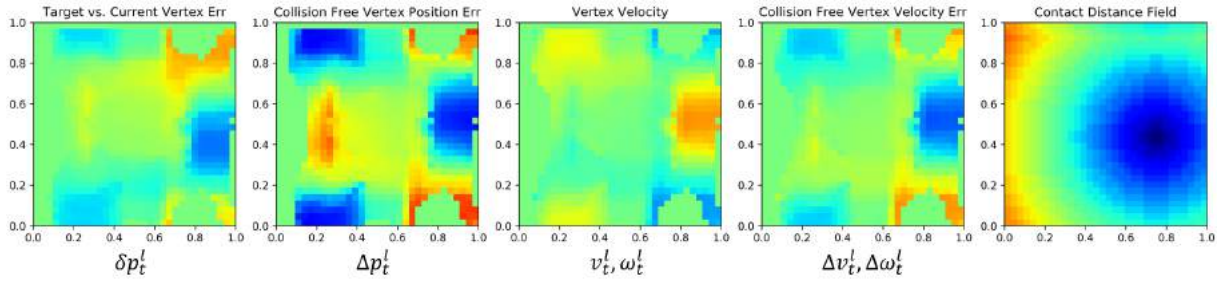

Fig. 4: Example visualization of features interpolated on the cylindrical projection (left 4) and contact distance field (right most) for link 5 on the Panda arm. For the feature images, blue means negative values, green means 0, and red means positive values. For the contact distance field, blue means 0, and red means the maximum possible distance on the image. Note the wrap around in the vertical axis of the contact distance field - this is because the vertical axis corresponds to the angular coordinates of the cylindrical projection. For the feature images, pixels outside the valid interpolation region have been zeroed out (see blue shaded region in panel E of Figure 3).

All training data is generated with Nvidia Isaac Gym [1], a GPU accelerated robotics simulator [36]. We set the simulator $dt = 0.01$s, and each trajectory runs for 500 simulator steps, which corresponds to 5s. We collect an observation every 4 simulator steps (so the observations are collected at a frequency of $f = 25$Hz), so each trajectory contains 125 observations. In total we collect 2800 simulated trajectories. The collected trajectories are cleaned by removing all contacts that are less than 1N in magnitude and last less than 0.1s. This resulted in a dataset with about 2% of the samples having positive contacts.

B. Link Mesh Processing

See panels A, B, and C in Figure 3 for the mesh simplification pipeline. There are three sets of meshes used. The first are the *visual meshes* obtained from the official Franka repository [2]. These are used for rendering and visualization only. The second set are the *projection meshes*, which are used for cylindrical projections, have lower resolution than the visual meshes. They also have vertices that are more spatially uniform and triangles that are more equilateral. We produce projection meshes by running TetWild [37] on the visual meshes. Each projection mesh has about 2000 triangles. The third set are the *collision meshes*, which are used by the simulation to perform collision checking. Having even lower resolution than the projection meshes, they are produced by running mesh decimation functions from libigl [38] and MeshLab [39] on the projection meshes. Each collision mesh has about 1000 triangles.

C. Cylindrical Projections

See panels D, E, and F in Figure 3 for the cylindrical projection pipeline. In our approach, obtaining the cylindrical projection representation of a mesh surface means obtaining a mapping from each vertex on the projection mesh to a normalized 2D cylindrical coordinate system. First, we manually specify a linear-interpolated spline with keypoints that roughly follow the geometry of a link mesh. This is only done once per robot link and takes about 5 minutes. Then, we form a tube along the spline and project vertices on the link to the surface of this tube. This step can be thought of as contracting a "sleeve" around a robot link. After this step, each mesh vertex is assigned two coordinates: s and ϕ, where s denotes the location of the mesh vertex along the spline, and ϕ denote the angle of vertex along the tube circumference. Lastly, these (s, ϕ) coordinates are normalized into (u, v) coordinates, where we map $s \in [s_{\text{min}}, s_{\text{max}}] \rightarrow u \in [0, 1]$ and $\phi \in [0, 2\pi] \rightarrow v \in [0, 1]$. To map from a mesh vertex to the cylindrical coordinates, we simply read off its precomputed (u, v) values. To map from a point in the UV space back to a mesh vertex, we use the mesh vertex whose cylindrical coordinates are the closest to the query point in UV space.

We use the cylindrical projection for two purposes - encoding contact distance fields which our trained neural network tries to predict, and encoding link surface features which the neural network uses for prediction. See Figure 4 for examples of both. To generate the contact distance fields for training, we first find the UV coordinates of the mesh vertices that are in contact, and then generate a distance field image, where the value of each pixel is the distance to the closest contact point in UV space. We use 32×32 as the image resolution, and it spans the entire UV space of $[0, 1] \times [0, 1]$. Importantly, distance in the V-axis "wraps around" as it corresponds to angles on a cylinder. This representation is desirable, because it easily allows encoding multiple contact points, and it provides a smooth and dense supervision signal for training - even points that are not in contact have useful information about the neighboring points that are. The latter is especially important in reducing sample complexity and improving network performance.

There are 5 types of features encoded with the cylindrical projections: 1) difference between the current and target link poses δp_t^l, 2) link velocities v_t^l, ω_t^l, 3) difference between the current and collision-free link poses Δp_t^l, 4) difference between the current and collision-free link velocities Δv_t^l, $\Delta \omega_t^l$, and 5) a mask that represents valid feature interpolation regions in the cylindrical projections.

First, we compute the vertex-wise versions of each of these features. For 1) and 3), this is done by taking the L2 norm of the appropriate vertex locations on the mesh surfaces, transformed by the link poses. For 2) and 4), this is done by converting linear and angular link velocities into linear vertex velocities: $v_t^l + d^i \otimes \omega_t^l$, where d^i is the difference

[1] https://developer.nvidia.com/isaac-gym
[2] https://github.com/frankaemika/franka_ros

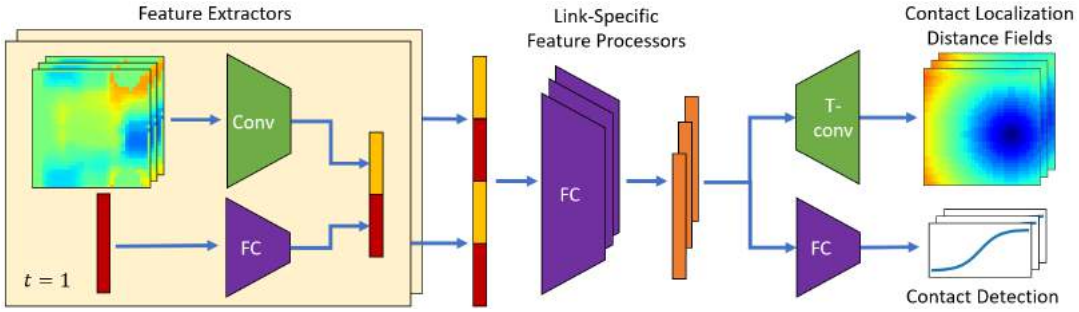

Fig. 5: Contact Localization Neural Network Architecture. The figure visualizes the case for 3 robot links (denoted by the elements that are repeated thrice) and an observation window of 2 timesteps (denote by the 2 repeated light yellow groups on the left). In practice we observe features and make predictions for 7 robot links across a window of 5 timesteps. Conv means convolution layers, FC means fully-connected, and T-conv means transposed convolutions. Note that for the inputs to the convolution layers, each square represents feature maps for *one* robot link, which by itself contains 5 channels.

between the ith vertex location and the link's center of mass, $\otimes$ denote the cross product, and angular velocities ω are expressed with the center of mass as the rotation origin. For the four of these features, the vertex-wise variants are vectors in $\mathbb{R}^3$. Second, we take the dot product between the vertex-wise features and each vertex's normal. Doing this reduces the dimension of vertex-wise features from 3 to 1, and it also ensures that they are represented relative to the robot arm's current frame, and not the world frame. Third, we form a feature image by placing scalar feature values of each vertex onto the corresponding UV coordinates and performing linear interpolation via Delaunay triangles to fill in the rest of the image. Like the distance field images, these feature images are also 32×32. Lastly, because not all regions in the UV space correspond to valid vertices on the mesh, we form a mask of valid interpolation regions and zero-out the pixels that are not in the mask. This mask is also given to the network as an additional channel in the input observations. We use alpha shapes [3] to produce *concave* hulls around the projected mesh vertices and use them as the masks. See the blue shaded region in panel F of Figure 3 for an illustration of the valid interpolation area.

D. Neural Network Model

See Figure 5 for a visualization of the network architecture. The contact localization model takes in a window of 5 past observations and predicts the contact state of all robot links at the latest time step in the window. There are 3 main modules in the network: a feature extractor, link-specific feature processors, and contact prediction heads. The feature extractor has 2 components - a convolutional encoder that processes all the feature images (Figure 4) and a fully-connected submodule to process the non-image observations. Non-image observations include joint velocities and the difference between the robot's joint angles and those of the robot in the "collision-free" simulation. The feature extractor independently processes observations from all links and across all timesteps in the window, concatenating the results into a latent vector. There are 7 link-specific feature processors that extract embeddings for each link from the shared latent vector. Lastly, the contact prediction heads take each link-specific embedding and make the appropriate predictions. There are two prediction heads: a fully-connected submodule that detects whether or not a link is in contact b^l and a transposed convolution decoder that predicts the contact distance fields to localize detected contacts.

Both the feature extractor and the contact prediction heads are shared across all links, and only the intermediate feature processors are link-specific. This weight sharing introduces an inductive bias that enables efficient network training.

The loss function is a weighted combination of a mean squared error loss for the contact distance field and a binary cross-entropy loss for contact detections. Because the dataset is heavily imbalanced (only 2% positive contacts), the binary cross-entropy loss weighs positive to negative samples with a $50:1$ ratio. 85% of the generated trajectories are used in the training set, with the remaining as the validation set.

To convert the model predictions into contact locations, we first check to see if the contact probability of each arm is over a threshold. If it is, then we extract contact points from the predicted contact distance fields. The value of each pixel in the distance field corresponds to the distance to the nearest contact point in pixel-space, so localizing contacts means finding pixels with zero or near-zero values. Here, we include all pixels below a threshold as the predicted contact points, then we map those pixel coordinates back to points on the mesh surface as the predicted contact locations.

IV. EXPERIMENTS

We performed two experiments to evaluate our proposed approach. The first compares the performance of the model that uses transposed convolution to predict contact distance fields (**CDF**) with that of a model which directly classifies (**CLS**) dense contact locations. The second demonstrates applying the trained networks to an obstacle mapping task. CLS has the same inputs and architecture as CDF, except its output heads, which are replaced with one that directly performs multi-label multi-class classification. This allows CLS to predict multiple contacts at the same time. Each class is a vertex of the projection meshes. Similar classifiers are used by multiple prior works, but our model classifies a total of 4077 contact points, which is more fine-grained than the 661 points used by [27] and the 20 by [29]; both works also use joint torque features, which our model does not.

[3] https://github.com/bellockk/alphashape

	Contact Detection			Contact Localization	
	ACC	FNR	FPR	AMCD-GT	AMCD-P
CLS	**94.6%**	31.9%	**4.4%**	**0.4** (1.7)	5.5 (4.6)
CDF	91.9%	**10.5%**	7.8%	3.6 (3.8)	**2.3** (3.4)
CLS-RF	94.1%	62.8%	**4.0%**	0.7 (2.0)	3.8 (3.9)
CDF-RF	78.6%	11.42%	21.3%	4.9 (4.4)	2.8 (3.6)

TABLE I: Contact detection and localization results with reduced-feature ablations. AMCD units are in cm. Parentheses refer to standard deviation.

A. Contact Localization

Two sets of metrics are used to compare contact prediction performance - one for detecting and one for localizing contacts. For contact detection, we report the accuracy (**ACC**), the false negative rate (**FNR**), and the false positive rate (**FPR**). Positive means there is a contact. For evaluating contact localizations, we compute the Average Minimum Contact Distance (AMCD). Denote a set of ground-truth contact locations on a link expressed in Cartesian space as $[c_1^l, \ldots, c_N^l]$ and the set of predicted contact locations on the same link as $[\hat{c}_1^l, \ldots, \hat{c}_M^l]$. There are two variants of AMCD, computed as follows: AMCD-GT $= \frac{1}{N}\sum_{n=1}^{N} \min_m \|c_n^l - \hat{c}_m^l\|_2$. AMCD-P $= \frac{1}{M}\sum_{n=1}^{M} \min_n \|\hat{c}_m^l - c_n^l\|_2$. A low AMCD-GT means that most ground truth contacts have predicted contacts that are nearby; a low AMCD-P means that most predicted contacts are near ground truth contacts. AMCD metrics can only be computed for time steps when there are true positives. Both AMCD and the contact detection metrics need to be viewed together to evaluate model performance. We also perform an ablation study by removing the image-based features from both CLS and CDF, resulting in models with reduced features, CLS-RF and CDF-RF.

See Table I for results computed for a validation dataset in simulation. Although CLS has higher ACC and and lower FPR, its FNR is $3\times$ that of CDF. For localization, CLS has a low AMCD-GT of 0.4cm, meaning ground truth contacts points mostly have predicted contact points nearby. However, it has a much higher AMCD-P of 5.5cm, meaning many of its predicted contact points are far away from ground truth contact points. By contrast, CDF has similar performance for both AMCD-GT and AMCD-P, achieving 3.6cm and 2.3cm respectively. Because CDF uses transposed convolution layers to predict a distance field, its predicted contacts tend to form clusters, so its errors are more spatially correlated. Taking the average of the two AMCDs, both CLS and CDF achieve a mean AMCD of 3.0cm. While the AMCD degradation of CLS-RF and CDF-RF are apparent but not significant, their contact detection metrics significantly deteriorate — CLS-RF has double the FNR as CLS, and CDF-RF has almost triple the FPR as CDF.

B. Obstacle Mapping

We apply the contact prediction models in an obstacle mapping task in both simulation and real world. Obstacles are modeled with voxel grids with 2cm resolution. Voxel values correspond to the probability that a voxel is occupied. All voxels have initial values of $\frac{1}{K}$, where K is the average number of occupied voxels in the training environments. To map obstacles, the robot performs a predefined but noisy exploration trajectory. Contact predictions are used to update the voxels with a Bayesian filter, which treats the probability of occupancy for each voxel to be independent from each other. We evaluate the negative log-likelihood (NLL), ACC, FNR, and FPR of the voxels in the volume swept by the robot. We also evaluate a variant of AMCD - the Average Minimum Voxel Distance (**AMVD**), which computes distances among predicted and ground-truth occupied voxels. Note these metrics are not about the contact locations on the robot arm, for which we do not have real-world ground truth labels. Rather, they are for the voxel occupancies of obstacles, which we manually measured in real-world experiments. Simulation results are aggregated over 200 noisy exploration trajectories on randomly generated obstacle environments not in the training set. Real world results use 10 on one obstacle configuration, also not in the training set. We control the Franka in the real world with [40].

	Simulation		Real World	
	CLS	CDF	CLS	CDF
ACC	95.4%	95.8%	96.3%	95.9%
FNR	70.3%	70.7%	70.5%	56.3%
FPR	3.1%	2.5%	0.6%	1.2%
NLL	0.26 (0.54)	0.20 (0.24)	0.29 (0.06)	0.27 (0.06)
AMVD-GT	11.6 (9.4)	10.5 (8.7)	6.5 (3.5)	4.6 (1.9)
AMVD-P	6.3 (5.9)	6.1 (5.2)	1.8 (1.7)	1.7 (1.3)

TABLE II: Voxel Mapping Results. AMVD units are in cm. Parentheses refer to standard deviation.

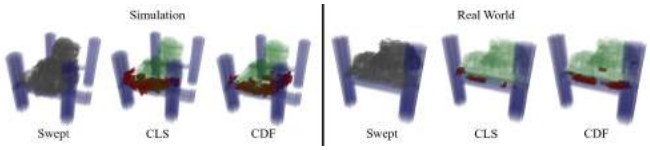

Fig. 6: Mapped Obstacles in Simulation and Real World. The gray voxels in "swept" are volumes where the robot had explored. For all figures, blue are ground truth occupied voxels, red the ones predicted to be occupied, and green predicted to be free.

See Table II for quantitative results and Figure 6 for a visualization of the mapped voxels. CLS and CDF achieve comparable performances in both simulation and the real world. The high voxel FNR is due to the small number of true positives in the volume swept by the robot. Voxel visualizations show that CDF has less false negative voxels, but slightly more false positive voxels, than CLS. This is in line with the contact detection results in Table I.

V. CONCLUSION

We train a neural network to detect and localize contacts on the surface of a 7-DoF robot arm. This is done while the robot is moving and without joint torque sensing, relaxing assumptions made in prior works. A novel cylindrical projection scheme is used to encode features and contact points on mesh surfaces. The network is trained with domain randomized data in simulation, and we demonstrate its use in an obstacle mapping task in both simulation and real world.

VI. ACKNOWLEDGMENT

The authors thank Saumya Saxena, Brian Okorn, and Tanya Marwah for their insightful discussions. This work is supported by NSF Grants No. DGE 1745016, IIS-1956163, and CMMI-1925130, the ONR Grant No. N00014-18-1-2775, ARL grant W911NF-18-2-0218 as part of the A2I2 program, and Nvidia NVAIL.

REFERENCES

[1] Q. Li, O. Kroemer, Z. Su, F. F. Veiga, M. Kaboli, and H. J. Ritter, "A review of tactile information: Perception and action through touch," *IEEE Transactions on Robotics*, 2020.

[2] M. C. Koval, M. Klingensmith, S. S. Srinivasa, N. S. Pollard, and M. Kaess, "The manifold particle filter for state estimation on high-dimensional implicit manifolds."

[3] M. Bauza, O. Canal, and A. Rodriguez, "Tactile mapping and localization from high-resolution tactile imprints."

[4] J. Liang, A. Handa, K. Van Wyk, V. Makoviychuk, O. Kroemer, and D. Fox, "In-hand object pose tracking via contact feedback and gpu-accelerated robotic simulation," *2020 IEEE International Conference on Robotics and Automation*, 2020.

[5] S. Wang, J. Wu, X. Sun, W. Yuan, W. T. Freeman, J. B. Tenenbaum, and E. H. Adelson, "3d shape perception from monocular vision, touch, and shape priors," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[6] J. Jung *et al.*, "Active shape completion using tactile glances," 2019.

[7] S. Ottenhaus, D. Renninghoff, R. Grimm, F. Ferreira, and T. Asfour, "Visuo-haptic grasping of unknown objects based on gaussian process implicit surfaces and deep learning."

[8] T. Matsubara and K. Shibata, "Active tactile exploration with uncertainty and travel cost for fast shape estimation of unknown objects," *Robotics and Autonomous Systems*, vol. 91, pp. 314–326, 2017.

[9] S. Ottenhaus, L. Kaul, N. Vahrenkamp, and T. Asfour, "Active tactile exploration based on cost-aware information gain maximization," *International Journal of Humanoid Robotics*, vol. 15, no. 01, p. 1850015, 2018.

[10] B. Saund, S. Choudhury, S. Srinivasa, and D. Berenson, "The blindfolded robot: A bayesian approach to planning with contact feedback," in *International Symposium on Robotics Research (ISRR)*, 2019.

[11] F. J. A. Chavez, J. Kangro, S. Traversaro, F. Nori, and D. Pucci, "Contact force and joint torque estimation using skin," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3900–3907, 2018.

[12] Y. Hirai, Y. Suzuki, T. Tsuji, and T. Watanabe, "Tough, bendable and stretchable tactile sensors array for covering robot surfaces," in *2018 IEEE International Conference on Soft Robotics (RoboSoft)*. IEEE, 2018, pp. 276–281.

[13] P. Mittendorfer, E. Yoshida, and G. Cheng, "Realizing whole-body tactile interactions with a self-organizing, multi-modal artificial skin on a humanoid robot," *Advanced Robotics*, vol. 29, no. 1, pp. 51–67, 2015.

[14] T. Bhattacharjee, A. Jain, S. Vaish, M. D. Killpack, and C. C. Kemp, "Tactile sensing over articulated joints with stretchable sensors," in *2013 World Haptics Conference (WHC)*. IEEE, 2013, pp. 103–108.

[15] E. Dean-Leon, J. R. Guadarrama-Olvera, F. Bergner, and G. Cheng, "Whole-body active compliance control for humanoid robots with robot skin," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 5404–5410.

[16] E. Dean-Leon, B. Pierce, F. Bergner, P. Mittendorfer, K. Ramirez-Amaro, W. Burger, and G. Cheng, "Tomm: Tactile omnidirectional mobile manipulator," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 2441–2447.

[17] G. Pang, J. Deng, F. Wang, J. Zhang, Z. Pang, and G. Yang, "Development of flexible robot skin for safe and natural human–robot collaboration," *Micromachines*, vol. 9, no. 11, p. 576, 2018.

[18] J. Wade, T. Bhattacharjee, R. D. Williams, and C. C. Kemp, "A force and thermal sensing skin for robots in human environments," *Robotics and Autonomous Systems*, vol. 96, pp. 1–14, 2017.

[19] M. Kaboli and G. Cheng, "Robust tactile descriptors for discriminating objects from textural properties via artificial robotic skin," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 985–1003, 2018.

[20] M. Kaboli, D. Feng, and G. Cheng, "Active tactile transfer learning for object discrimination in an unstructured environment using multimodal robotic skin," *International Journal of Humanoid Robotics*, vol. 15, no. 01, p. 1850001, 2018.

[21] T. Bhattacharjee, P. M. Grice, A. Kapusta, M. D. Killpack, D. Park, and C. C. Kemp, "A robotic system for reaching in dense clutter that integrates model predictive control, learning, haptic mapping, and planning." Georgia Institute of Technology, 2014.

[22] G. Cheng, E. Dean-Leon, F. Bergner, J. R. G. Olvera, Q. Leboutet, and P. Mittendorfer, "A comprehensive realization of robot skin: Sensors, sensing, control, and applications," *Proceedings of the IEEE*, vol. 107, no. 10, pp. 2034–2051, 2019.

[23] L. Manuelli and R. Tedrake, "Localizing external contact using proprioceptive sensors: The contact particle filter," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 5062–5069.

[24] M. Benallegue, P. Gergondet, H. Audrerr, A. Mifsud, M. Morisawa, F. Lamiraux, A. Kheddar, and F. Kanehiro, "Model-based external force/moment estimation for humanoid robots with no torque measurement," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 3122–3129.

[25] A. Zwiener, R. Hanten, C. Schulz, and A. Zell, "Armcl: Arm contact point localization via monte carlo localization." in *IROS*, 2019, pp. 7105–7111.

[26] S. Wang, A. Bhatia, M. T. Mason, and A. M. Johnson, "Contact localization using velocity constraints," *International Conference on Intelligent Robots and Systems*, 2020.

[27] A. Zwiener, C. Geckeler, and A. Zell, "Contact point localization for articulated manipulators with proprioceptive sensors and machine learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 323–329.

[28] D. Popov and A. Klimchik, "Real-time external contact force estimation and localization for collaborative robot," in *2019 IEEE International Conference on Mechatronics (ICM)*, vol. 1. IEEE, 2019, pp. 646–651.

[29] ——, "Transfer learning for collision localization in collaborative robotics," in *Proceedings of the 3rd International Conference on Applications of Intelligent Systems*, 2020, pp. 1–7.

[30] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 23–30.

[31] K. Narukawa, T. Yoshiike, K. Tanaka, and M. Kuroda, "Real-time collision detection based on one class svm for safe movement of humanoid robot," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. IEEE, 2017, pp. 791–796.

[32] G. Bledt, P. M. Wensing, S. Ingersoll, and S. Kim, "Contact model fusion for event-based locomotion in unstructured terrains," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–8.

[33] N. Rotella, S. Schaal, and L. Righetti, "Unsupervised contact learning for humanoid estimation and control," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 411–417.

[34] G. Cioffi, S. Klose, and A. Wahrburg, "Data-efficient online classification of human-robot contact situations," in *2020 European Control Conference (ECC)*. IEEE, 2020, pp. 608–614.

[35] C. Gaz, M. Cognetti, A. Oliva, P. R. Giordano, and A. De Luca, "Dynamic identification of the franka emika panda robot with retrieval of feasible parameters using penalty-based optimization," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4147–4154, 2019.

[36] J. Liang, V. Makoviychuk, A. Handa, N. Chentanez, M. Macklin, and D. Fox, "Gpu-accelerated robotic simulation for distributed reinforcement learning," *Conference on Robot Learning*, 2018.

[37] Y. Hu, Q. Zhou, X. Gao, A. Jacobson, D. Zorin, and D. Panozzo, "Tetrahedral meshing in the wild," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 60:1–60:14, July 2018. [Online]. Available: http://doi.acm.org/10.1145/3197517.3201353

[38] A. Jacobson, D. Panozzo, *et al.*, "libigl: A simple C++ geometry processing library," 2018, https://libigl.github.io/.

[39] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, "Meshlab: an open-source mesh processing tool." in *Eurographics Italian chapter conference*, vol. 2008. Salerno, 2008, pp. 129–136.

[40] K. Zhang, M. Sharma, J. Liang, and O. Kroemer, "A modular robotic arm control stack for research: Franka-interface and frankapy," *arXiv preprint arXiv:2011.02398*, 2020.

[41] W. Falcon, "Pytorch lightning," *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, vol. 3, 2019.

APPENDIX I
DATA COLLECTION

A. Franka Links

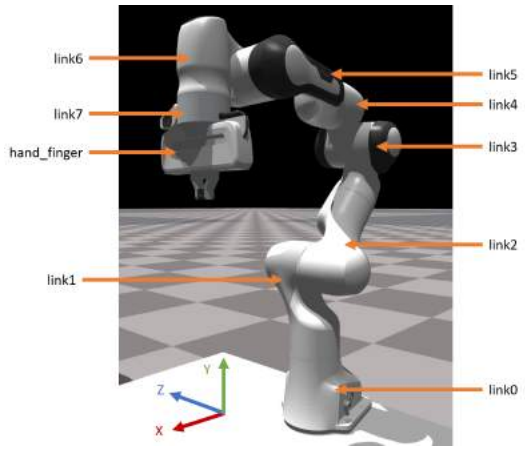

Fig. 7: Franka Link Names and Coordinate Axes. We collect observations and make predictions for the last 7 links on the Franka arm, from link2 to hand_finger, inclusively. Franka grippers are closed, and we combine it with the rest of the hand to form the one hand_finger link mesh.

See Figure 7 for the Franka links our trained contact sensing models operate on, and see Figure 8 for visualizations of cylindrical projections for all 7 links.

B. Obstacle Placement

We generate two sets of random obstacles in the scene. Each set contains 3 bars with square cross sections of width 4cm. Out of the 3 bars, 2 are vertical, and 1 is horizontal and attached to the vertical bars. For the vertical bars, we sample an X position uniformly in the range of $[13, 60]$cm in front of the robot. For the Z-axis, one vertical bar samples from $[15, 30]$cm, and the other $[-15, -30]$cm, placing the two bars at both sides of the robot. Once the positions of the vertical bars are determined, we sample two Y coordinates from the range $[20, 50]$cm. These values are then used as the points on the vertical bars at which we attach the horizontal bar.

C. Exploration Policies

For each trajectory, we first sample an initial joint angle configuration within the range $[q_0 - \Delta q, q_0 + \Delta q]$, with q_0 being the home configuration seen in Figure 7, and $\Delta q = [0, 0, 0, 5°, 5°, 10°, 20°]$. For informed exploration, the robot executes a rectangle-shaped trajectory near where obstacles are typically generated. Waypoints along the rectangle are perturbed with uniformly sampled noise in $[-3, 3]$cm. For random exploration, while the trajectory remains under the time horizon of 500 steps, we sample new delta goal end-effector poses for the robot to reach. For the translation component of the delta pose, we sample the direction and magnitude separately. Direction is sampled from a discrete distribution with the following probabilities:

$$+X : 0.15, +Y : 0.3, +Z : 0.1$$
$$-X : 0.05, -Y : 0.3, -Z : 0.1$$

Magnitude is sampled uniformly from the range $[10, 20]$cm. For the rotation component of the delta pose, we uniformly sample delta euler angles with the range $[0°, 20°]$. Finally, we uniformly sample a time horizon in the range $[20, 40]$ steps, during which the delta pose command is completed with min-jerk interpolation.

D. Domain Randomization

In addition to randomizing obstacle placements, we also randomize the inertial parameters of the robot and the gains for the impedance control. For the inertial parameters, we uniformly sample a mass offset in the range of $[-0.5, 0.5]$kg and and center-of-mass offset in $[-1, 1]$cm, which are added to the base values obtained in [35].

For the impedance gains, we uniformly sample translation gains K_T in the range $[200, 2000]$ and rotation gains K_R in the range $[3, 6]$. To use these values, let $K = diag([K_T, K_T, K_T, K_R, K_R, K_R])$ be the 6×6 diagonal gains matrix and J be the 6×7 analytical Jacobian for the robot end-effector that encodes rotations as euler angles. Then, given a delta pose p_d, the commanded torque is $\tau = J^\top(-Kp_d - DJ\dot{q})$, where D is the damping term $D = 2\sqrt{K}$, and $\dot{q}$ is the joint velocity vector. For simplicity, we have left out the terms that correspond to gravity compensation and Coriolis forces.

E. Data Statistics

Out of the 2800 unique trajectories generated, 2400 were used in the training set. Each trajectory contains 25 non-overlapping observation windows of 7 links, bringing the total amount of training data samples to $2400 \times 5 \times 7 = 420000$. Out of these, 8358 samples have at least one positive contact, so there are 1.99% positive contacts in the training dataset. In addition, 2209 samples have more than one positive contacts, so there are 0.53% of samples with multiple contacts.

APPENDIX II
NEURAL NETWORK MODEL

Here we detail the neural network architecture From Figure 5, the Fully-Connected (FC) module of the feature extractor is a multilayer perceptron (MLP) with 3 hidden layers of sizes $[128, 128, 64]$. The convolution layers channels $[5, 16, 32, 32]$, each with kernel size 5 and a stride of 2. Each link-specific feature processor is an MLP with hidden layers $[128, 128, 64]$. The contact distance field decoder has transposed convolution layers with channels $[64, 32, 32, 1]$, kernel sizes $[5, 6, 6]$, and strides of 2. The contact detection module is an MLP with hidden layers $[32, 32]$. We use Leaky-ReLU as the nonlinearities. Network is implemented with PyTorch Lightning [41], using the Adam optimizer with a batch size of 256 and an initial learning rate $1e-3$.

APPENDIX III
CONTACT PREDICTIONS

See Figure 9 for contact detection visualizations with both CLS and CDF, and Figure 10 for contact localization predictions in the cylindrical projection coordinates.

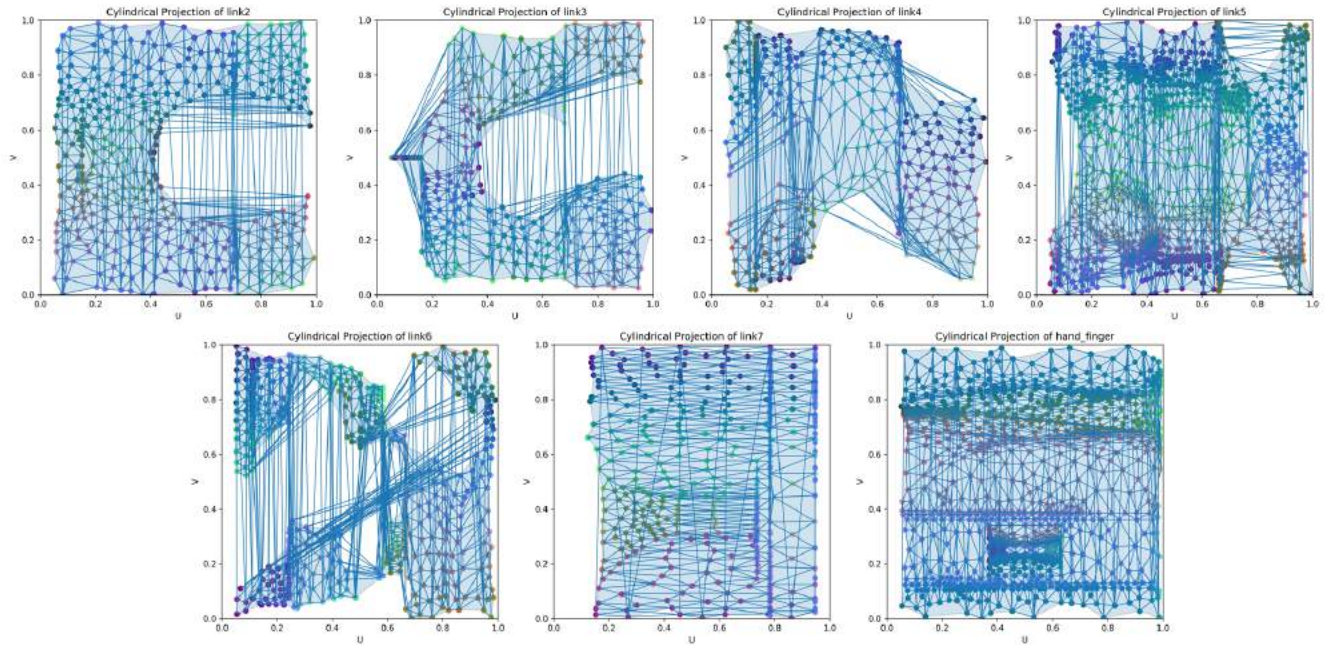

Fig. 8: Cylindrical Projections for All Links. Cylindrical coordinates are normalized in the range of $[0, 1]$. The horizontal U-axis denotes the coordinate along the tube, and vertical V-axis denotes the angular coordinate along the tube's circumference. Each dot is a vertex, and the color of the dot indicates its normal direction. Edges are mesh triangle edges. Blue shaded region denotes the concave hull used to define valid projection interpolation region.

Fig. 9: Contact Detection Visualizations on Test Data. Each column is the execution of one trajectory. The top row have plots corresponding to predictions made by CLS, and the bottom CDF. In each plot, link-wise ground truth and predicted binary contact detections are visualized, with purple representing negative contacts, and yellow positive contacts. The X-axis represents prediction time segments, so each unit corresponds to 5 observations, one made every 4 simulation steps. In general, CLS has more FNs, and CDF has more FPs, which is consistent with results in Table I.

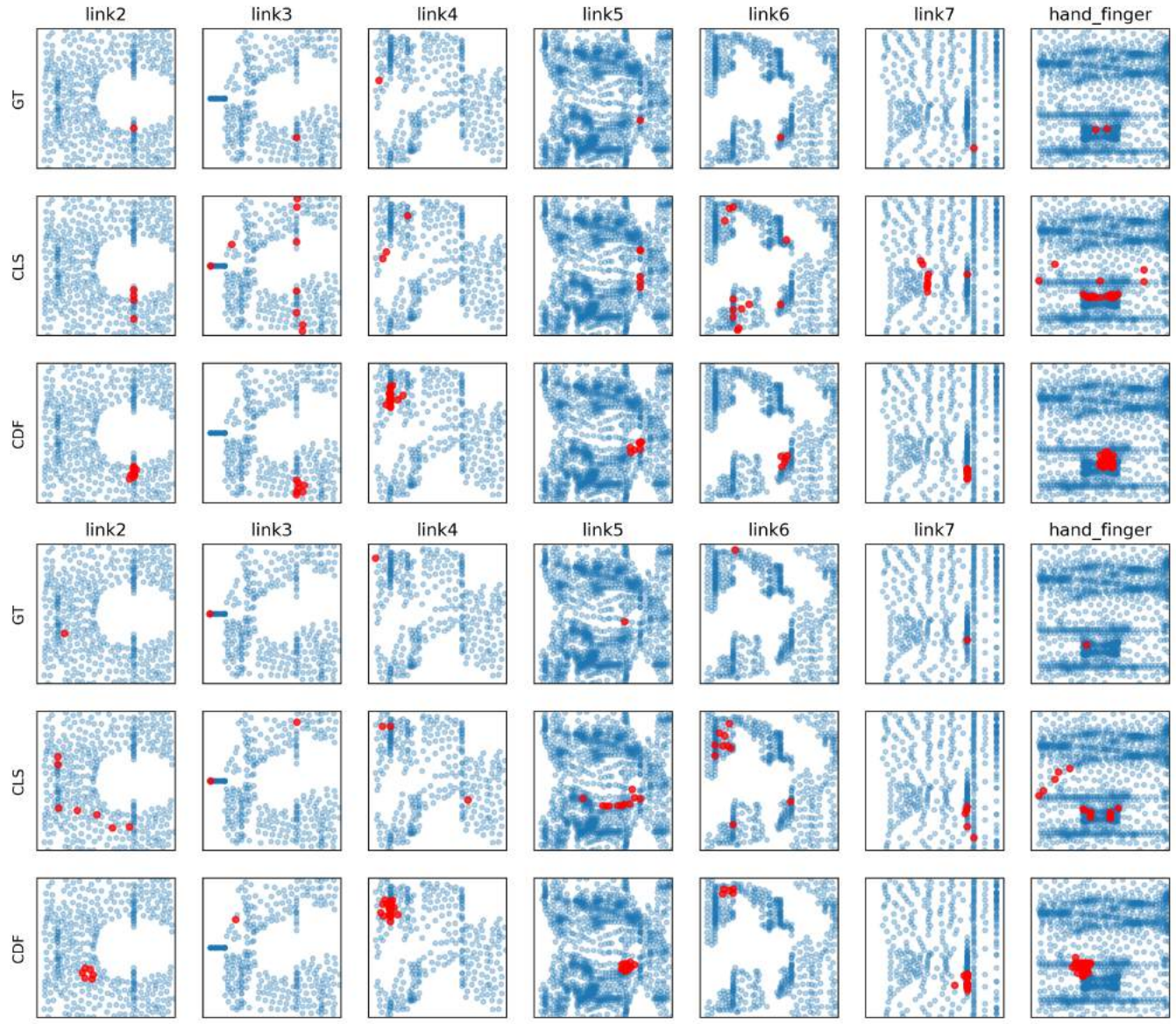

Fig. 10: Contact Localization Visualizations on Test Data. We visualize two sets (top 3 and bottom 3 rows) of examples of predicted contact localizations in cylindrical coordinates. Blue points are projection mesh vertices; red points are contact locations. Each column is a positive sample for a link. For each set, the top row are ground truth contact points, middle are predicted by CLS, and bottom by CDF. Due to its use of distance fields, CDF's predicted contacts tend to be less scattered than those of CLS.