Robobarista: Learning to Manipulate Novel Objects via Deep Multimodal Embedding

Jaeyong Sung, Seok Hyun Jin, Ian Lenz, and Ashutosh Saxena

Abstract— There is a large variety of objects and appliances in human environments, such as stoves, coffee dispensers, juice extractors, and so on. It is challenging for a roboticist to program a robot for each of these object types and for each of their instantiations. In this work, we present a novel approach to manipulation planning based on the idea that many household objects share similarly-operated object parts. We formulate the manipulation planning as a structured prediction problem and learn to transfer manipulation strategy across different objects by embedding point-cloud, natural language, and manipulation trajectory data into a shared embedding space using a deep neural network. In order to learn semantically meaningful spaces throughout our network, we introduce a method for pre-training its lower layers for multimodal feature embedding and a method for fine-tuning this embedding space using a loss-based margin. In order to collect a large number of manipulation demonstrations for different objects, we develop a new crowd-sourcing platform called Robobarista. We test our model on our dataset consisting of 116 objects and appliances with 249 parts along with 250 language instructions, for which there are 1225 crowd-sourced manipulation demonstrations. We further show that our robot with our model can even prepare a cup of a latte with appliances it has never seen before.[1]

I. INTRODUCTION

Consider the espresso machine in Figure 1 — even without having seen this machine before, a person can prepare a cup of latte by visually observing the machine and reading an instruction manual. This is possible because humans have vast prior experience with manipulating differently-shaped objects that share common parts such as 'handles' and 'knobs.' In this work, our goal is to give robots the same capabilites – specifically, to enable robots to generalize their manipulation ability to novel objects and tasks (e.g. toaster, sink, water fountain, toilet, soda dispenser, etc.). We build an algorithm that uses a large knowledge base of manipulation demonstrations to infer an appropriate manipulation trajectory for a given pair of point-cloud and natural language instructions.

If the robot's sole task is to manipulate one specific espresso machine or just a few types of 'handles', a roboticist could manually program the exact sequence to be executed. However, human environments contain a huge variety of objects, which makes this approach un-scalable and infeasible. Classification of objects or object parts (e.g. 'handle') alone does not provide enough information for robots to actually manipulate them, since semantically-similar objects

Authors are with Department of Computer Science, Cornell University. Email: {jysung,sj372,ianlenz,asaxena} @cs.cornell.edu Jaeyong Sung is also a visiting student researcher at Stanford University. Ashutosh Saxena is also with Brain of Things, Inc.

[1]Parts of this work were presented at ISRR 2015 (Sung et al. [61])
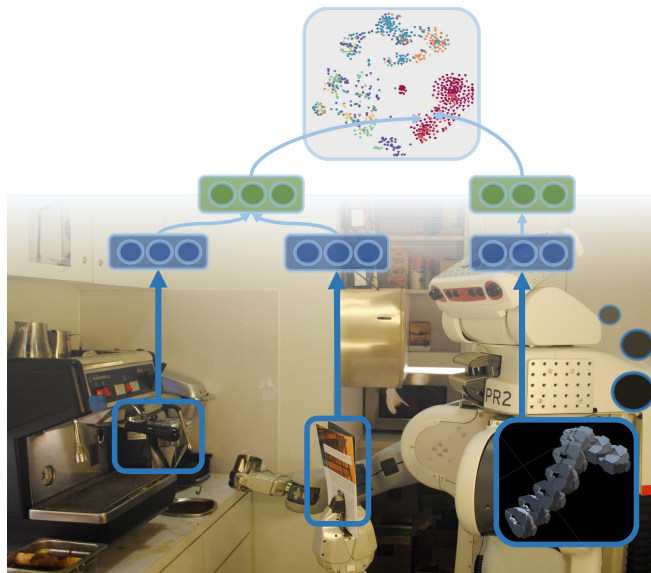

Fig. 1. **First encounter of an espresso machine** by our PR2 robot. Without ever having seen the machine before, given language instructions and a point-cloud from Kinect sensor, our robot is capable of finding appropriate manipulation trajectories from prior experience using our deep multimodal embedding model.

or parts might be operated completely differently – consider, for example, manipulating the 'handle' of a urinal, as opposed to a door 'handle'. Thus, rather than relying on scene understanding techniques [8, 39, 19], we directly use 3D point-clouds for manipulation planning using machine learning algorithms.

The key idea of our work is that objects designed for use by humans share many similarly-operated *object parts* such as 'handles', 'levers', 'triggers', and 'buttons'; thus, manipulation motions can be transferred even between completely different objects if we represent these motions with respect to these parts. For example, even if the robot has never seen an espresso machine before, it should be able to manipulate it if it has previously seen similarly-operated parts of other objects such as a urinal, soda dispenser, or restroom sink, as illustrated in Figure 2. Object parts that are operated in similar fashion may not necessarily carry the same part name (e.g. 'handle') but should have some similarity in their shapes that allows motions to be transferred between completely different objects.

Going beyond manipulation based on simple semantic classes is a significant challenge which we address in this work. Instead, we must also make use of visual information

(point-clouds), and a natural language instruction telling the robot what to do since many possible affordances can exist for the same part. Designing useful features for either of these modalities alone is already difficult, and designing features which combine the two for manipulation purposes is extremely challenging.

Obtaining a good common representation between different modalities is challenging for two main reasons. First, each modality might intrinsically have very different statistical properties – for example, most trajectory representations are inherently dense, while a bag-of-words representation of language is by nature sparse. This makes it challenging to apply algorithms designed for unimodal data, as one modality might overpower the others. Second, even with expert knowledge, it is extremely challenging to design joint features between such disparate modalities. Humans are able to map similar concepts from different sensory system to the same concept using *common representation* between different modalities [15]. For example, we are able to correlate the appearance with feel of a banana, or a language instruction with a real-world action. This ability to fuse information from different input modalities and map them to actions is extremely useful to a household robot.

In this work, we use deep neural networks to learn a shared embedding between the combination of object parts in the environment (point-cloud) and natural language instructions, and manipulation trajectories (Fig. 3). This means that all three modalities are projected to the *same* feature space. We introduce an algorithm that learns to pull semantically similar environment/language pairs and their corresponding trajectories to the same regions, and push environment/language pairs away from irrelevant trajectories based on how irrelevant these trajectories are. Our algorithm allows for efficient inference because, given a new instruction and point-cloud, we only need to find the nearest trajectory to the projection of this pair in the learned embedding space, which can be done using fast nearest-neighbor algorithms [45].

In the past, deep learning methods have shown impressive results for learning features in a wide variety of domains [33, 54, 21] and even learning cross-domain embeddings for, for example, language and image features [57]. In contrast to these existing methods, here we present a new pre-training algorithm for initializing networks to be used for joint embedding of different modalities.

Such deep learning algorithms require a large dataset for training. However, collecting a large enough dataset of expert demonstrations on a large number of objects is very expensive as it requires joint physical presence of the robot, an expert, and the object to be manipulated. In this work, we show that we can crowd-source the collection of manipulation demonstrations to the public over the web through our Robobarista platform. Since crowd-sourced demonstrations might be noisy and sub-optimal, we present a new learning algorithm which handles this noise. With our noise handling algorithm, our model trained with crowd-sourced demonstrations outperforms the model trained with expert demonstrations, even with the significant amount of

noise in crowd-sourced manipulation demonstrations.

Furthermore, in contrast to previous approaches based on learning from demonstration (LfD) that learn a mapping from a state to an action [4], our work complements LfD as we focus on the entire manipulation motion, as opposed to a sequential state-action mapping.

Our Robobarista web-based crowdsourcing platform (`http://robobarista.cs.cornell.edu`) allowed us to collect a large dataset of *116 objects* with *250 natural language instructions* for which there are *1225 crowd-sourced manipulation trajectories* from 71 non-expert users, which we use to validate our methods. We also present experiments on our robot using our approach. In summary, the key contributions of this work are:

- We present a novel approach to manipulation planning via *part-based transfer* between different objects that allows manipulation of novel objects.
- We introduce a new *deep learning model* that handles three modalities with noisy labels from crowd-sourcing.
- We introduce a new algorithm, which learns an *embedding space* while enforcing a varying and *loss-based margin*, along with a new unsupervised pre-training method which outperforms standard pre-training algorithms [20].
- We develop an online platform which allows the incorporation of *crowd-sourcing* to manipulation planning and introduces a large-scale manipulation dataset.
- We evaluate our algorithms on this dataset, showing significant improvement over other state-of-the-art methods.

II. RELATED WORK

Improving robotic perception and teaching manipulation strategies to robots has been a major research area in recent years. In this section, we describe related work in various aspects of learning to manipulate novel objects.

Scene Understanding. In recent years, there has been significant research focus on semantic scene understanding [39, 32, 33, 75], human activity detection [59, 25], and features for RGB-D images and point-clouds [55, 35]. Similar to our idea of using part-based transfers, the deformable part model [19] was effective in object detection. However, classification of objects, object parts, or human activities alone does not provide enough information for a robot to reliably plan manipulation. Even a simple category such as 'kitchen sinks' has a huge amount of variation in how different instances are manipulated – for example, handles and knobs must be manipulated differently, and different orientations and positions of these parts require very different strategies such as pulling the handle upwards, pushing upwards, pushing sideways, and so on. On the other hand, direct perception approaches [18, 34] skip the intermediate object labels and directly perceive affordances based on the shape of the object. These works focus on detecting the part known to afford certain actions, such as 'pour,' given the object, while we focus on predicting the correct motion given the object part.

Manipulation Strategy. Most works in robotic manipulation focus on task-specific manipulation of *known* objects—for example, baking cookies with known tools [9] and folding the laundry [42] – or focus on learning specific motions such as grasping [30] and opening doors [14]. Others [60, 43] focus on sequencing manipulation tasks assuming perfect manipulation primitives such as *grasp* and *pour* are available. Instead, here, we use learning to generalize to manipulating *novel* objects never seen before by the robot, without relying on preprogrammed motion primitives.

For the more general task of manipulating new instances of objects, previous approaches rely on finding articulation [58, 50] or using interaction [29], but they are limited by tracking performance of a vision algorithm. Many objects that humans operate daily have small parts such as 'knobs', which leads to significant occlusion as manipulation is demonstrated. Another approach using part-based transfer between objects has been shown to be successful for grasping [11, 13]. We extend this approach and introduce a deep learning model that enables part-based transfer of *trajectories* by automatically learning relevant features. Our focus is on the generalization of manipulation trajectory via part-based transfer using point-clouds without knowing objects a priori and without assuming any of the sub-steps ('approach', 'grasping', and 'manipulation').

A few recent works use deep learning approaches for robotic manipulation. Levine et al. [38] use a Gaussian mixture model to learn system dynamics, then use these to learn a manipulation policy using a deep network. Lenz et al. [37] use a deep network to learn system dynamics for real-time model-predictive control. Both these works focus on learning low-level controllers, whereas here we learn high-level manipulation trajectories.

Learning from Demonstration. Several successful approaches for teaching robots tasks, such as helicopter maneuvers [1] or table tennis [46], have been based on Learning from Demonstration (LfD) [4]. Although LfD allows end users to demonstrate a manipulation task by simply taking control of the robot's arms, it focuses on learning individual actions and separately relies on high level task composition [40, 12] or is often limited to previously seen objects [49, 48]. We believe that learning a single model for an action like 'turning on' is impossible because human environments have so many variations.

Unlike learning a model from demonstration, instance-based learning [2, 17] replicates one of the demonstrations. Similarly, we directly transfer one of the demonstrations, but focus on generalizing manipulation planning to completely new objects, enabling robots to manipulate objects they have never seen before.

Metric Embedding. Several works in machine learning make use of the power of shared embedding spaces. LMNN [72] learns a max-margin Mahalanobis distance for a uni-modal input feature space. Weston et al. [73] learn linear mappings from image and language features to a common embedding space for automatic image annotation. Moore et al. [44] learn to map songs and natural language tags to a shared embedding space. However, these approaches learn only a shallow, linear mapping from input features, whereas here we learn a deep non-linear mapping which is less sensitive to input representations.

Deep Learning. In recent years, deep learning algorithms have enjoyed huge successes, particularly in the domains of of vision and natural language processing (e.g. [33, 54]). In robotics, deep learning has previously been successfully used for detecting grasps for novel objects in multi-channel RGB-D images [36] and for classifying terrain from long-range vision [21].

Ngiam et al. [47] use deep learning to learn features incorporating both video and audio modalities. Sohn et al. [56] propose a new generative learning algorithm for multi-modal data which improves robustness to missing modalities at inference time. In these works, a single network takes all modalities as inputs, whereas here we perform joint embedding of multiple modalities using multiple networks.

Several previous works use deep networks for joint embedding between different feature spaces. Mikolov et al. [41] map different languages to a joint feature space for translation. Srivastava and Salakhutdinov [57] map images and natural language "tags" to the same space for automatic annotation and retrieval. While these works use conventional pre-training algorithms, here we present a new pre-training approach for learning embedding spaces and show that it outperforms these existing methods (Sec. VIII-C.) Our algorithm trains each layer to map similar cases to similar areas of its feature space, as opposed to other methods which either perform variational learning [22] or train for reconstruction [20].

Hu et al. [24] also use a deep network for metric learning for the task of face verification. Similar to LMNN [72], Hu et al. [24] enforces a constant margin between distances among inter-class objects and among intra-class objects. In Sec. VIII-C, we show that our approach, which uses a loss-dependent variable margin, produces better results for our problem. Our work builds on deep neural network to embed three different modalities of point-cloud, language, and trajectory into shared embedding space while handling lots of label-noise originating from crowd-sourcing.

Crowd-sourcing. Many approaches to teaching robots manipulation and other skills have relied on demonstrations by skilled experts [4, 1]. Among previous efforts to scale teaching to the crowd [10, 62, 27], Forbes et al. [17] employs a similar approach towards crowd-sourcing but collects multiple instances of similar table-top manipulation with same object. Others also build web-based platform for crowd-sourcing manipulation [65, 66]. However, these approaches either depend on the presence of an expert (due to required special software), or require a real robot at a remote location. Our Robobarista platform borrows some components of work from Alexander et al. [3], but works on any standard web browser with OpenGL support and incorporates real point-clouds of various scenes.
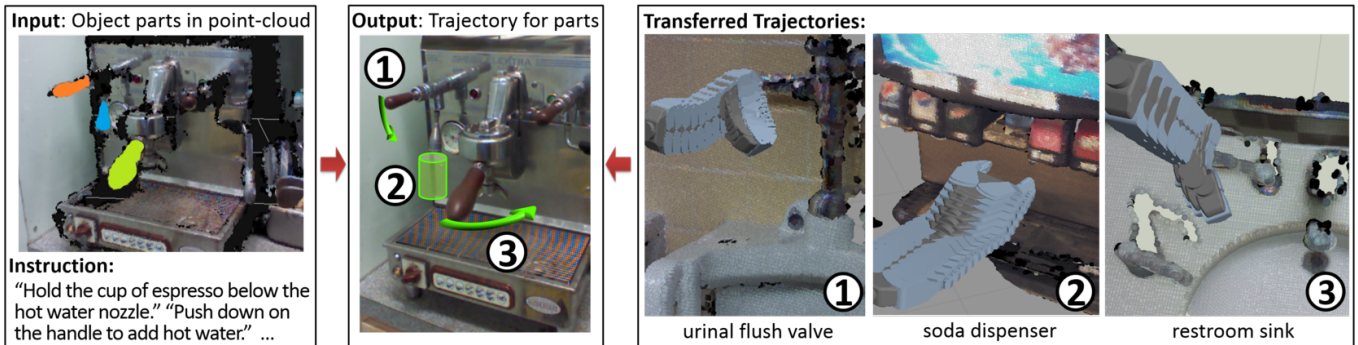
Fig. 2. **Mapping object part and natural language instruction input to manipulation trajectory output.** Objects such as the espresso machine consist of distinct object parts, each of which requires a distinct manipulation trajectory for manipulation. For each part of the machine, we can re-use a manipulation trajectory that was used for some other object with similar parts. So, for an object part in a point-cloud (each object part colored on left), we can find a trajectory used to manipulate some other object (labeled on the right) that can be *transferred* (labeled in the center). With this approach, a robot can operate a new and previously unobserved object such as the 'espresso machine', by successfully transferring trajectories from other completely different but previously observed objects. Note that the input point-cloud is very noisy and incomplete (black represents missing points).

III. OUR APPROACH

Our goal is to build an algorithm that allows a robot to infer a manipulation trajectory when it is introduced to a new object or appliance and its natural language instruction manual. The intuition for our approach is that many differently-shaped objects share similarly-operated object parts; thus, the manipulation trajectory of an object can be transferred to a completely different object if they share similarly-operated parts.

For example, the motion required to operate the handle of the espresso machine in Figure 2 is almost identical to the motion required to flush a urinal with a handle. By identifying and transferring trajectories from prior experience with parts of other objects, robots can manipulate even objects they have never seen before.

We first formulate this problem as a structured prediction problem as shown in Figure 2. Given a point-cloud for each part of an espresso machine and a natural language instruction such as 'Push down on the handle to add hot water', our algorithm outputs a trajectory which executes the task, using a pool of prior motion experience.

This is a challenging problem because the object is entirely new to the robot, and because it must jointly consider the point-cloud, natural language instruction, and each potential trajectory. Moreover, manually designing useful features from these three modalities is extremely challenging.

We introduce a *deep multimodal embedding* approach that learns a shared, semantically meaningful embedding space between these modalities, while dealing with a noise in crowd-sourced demonstration data. Then, we introduce our Robobarista crowd-sourcing platform, which allows us to easily scale the collection of manipulation demonstrations to non-experts on the web.

A. Problem Formulation

Our goal is to learn a function f that maps a given pair of point-cloud $p \in \mathcal{P}$ of an object part and a natural language

instruction $l \in \mathcal{L}$ to a trajectory $\tau \in \mathcal{T}$ that can manipulate the object part as described by free-form natural language l:

$$f : \mathcal{P} \times \mathcal{L} \to \mathcal{T} \qquad (1)$$

For instance, given the handle of the espresso machine in Figure 2 and an natural language instruction 'Push down on the handle to add hot water', the algorithm should output a manipulation trajectory that will correctly accomplish the task on the object part according to the instruction.

Point-cloud Representation. Each instance of a point-cloud $p \in \mathcal{P}$ is represented as a set of n points in three-dimensional Euclidean space where each point (x, y, z) is represented with its RGB color (r, g, b):

$$p = \{p^{(i)}\}_{i=1}^n = \{(x, y, z, r, g, b)^{(i)}\}_{i=1}^n$$

The size of this set varies for each instance. These points are often obtained by stitching together a sequence of sensor data from an RGBD sensor [26].

Trajectory Representation. Each trajectory $\tau \in \mathcal{T}$ is represented as a sequence of m *waypoints*, where each waypoint consists of gripper status g, translation (t_x, t_y, t_z), and rotation (r_x, r_y, r_z, r_w) with respect to the origin:

$$\tau = \{\tau^{(i)}\}_{i=1}^m = \{(g, t_x, t_y, t_z, r_x, r_y, r_z, r_w)^{(i)}\}_{i=1}^m$$

where $g \in \{$"open", "closed", "holding"$\}$. g depends on the type of the end-effector, which we have assumed to be a two-fingered parallel-plate gripper like that of PR2 or Baxter. The rotation is represented as quaternions (r_x, r_y, r_z, r_w) instead of the more compact Euler angles to prevent problems such as gimbal lock.

Smooth Trajectory. To acquire a smooth trajectory from a waypoint-based trajectory τ, we interpolate intermediate waypoints. Translation is linearly interpolated and the quaternion is interpolated using spherical linear interpolation (Slerp) [53].

B. Data Pre-processing

Each of the point-cloud, language, and trajectory (p, l, τ) can have any length. Thus, we fit raw data from each modality into a fixed-length vector.

We represent point-cloud p of any arbitrary length as an occupancy grid where each cell indicates whether any point lives in the space it represents. Because point-cloud p consists of only the part of an object which is limited in size, we can represent p using two occupancy grid-like structures of size $10 \times 10 \times 10$ voxels with different scales: one with each voxel spanning a cube of $1 \times 1 \times 1 (cm)$ and the other with each cell representing $2.5 \times 2.5 \times 2.5 (cm)$.

Each language instruction is represented as a fixed-size bag-of-words representation with stop words removed. Finally, for each trajectory $\tau \in \mathcal{T}$, we first compute its smooth interpolated trajectory $\tau_s \in \mathcal{T}_s$ (Sec. III-A), and then normalize all trajectories $\mathcal{T}_s$ to the same length while preserving the sequence of gripper states such as 'opening', 'closing', and 'holding'.

C. Direct manipulation trajectory transfer

Even if we have a trajectory to transfer, a conceptually transferable trajectory is not necessarily directly compatible if it is represented with respect to an inconsistent reference point.

To make a trajectory compatible with a new situation without modifying the trajectory, we need a representation method for trajectories, based on point-cloud information, that allows a *direct transfer of a trajectory without any modification*.

Challenges. Making a trajectory compatible when transferred to a different object or to a different instance of the same object without modification can be challenging depending on the representation of trajectories and the variations in the location of the object, given in point-clouds.

Many approaches which control high degree of freedom arms such as those of PR2 or Baxter use configuration-space trajectories, which store a time-parameterized series of joint angles [64]. While such approaches allow for direct control of joint angles during control, they require costly recomputation for even a small change in an object's position or orientation.

One approach that allows execution without modification is representing trajectories with respect to the object by aligning via point-cloud registration (e.g. [17]). However, a large object such as a stove might have many parts (e.g. knobs and handles) whose positions might vary between different stoves. Thus, object-aligned manipulation of these parts would not be robust to different stoves, and in general would impede transfer between different instances of the same object.

Lastly, it is even more challenging if two objects require similar trajectories, but have slightly different shapes. And this is made more difficult by limitations of the point-cloud data. As shown in left of Fig. 2, the point-cloud data, even when stitched from multiple angles, are very noisy compared to the RGB images.
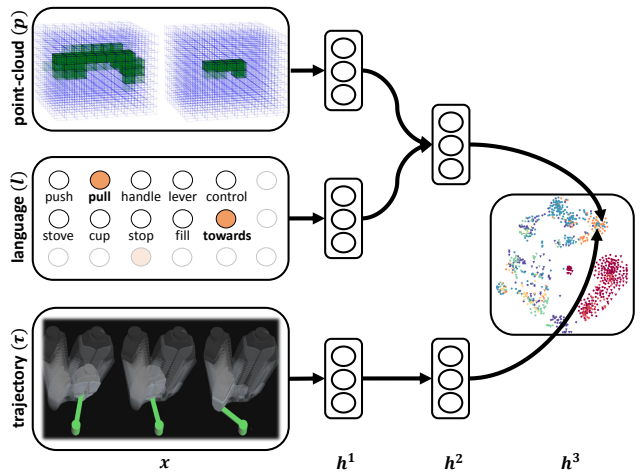

Fig. 3. **Deep Multimodal Embedding:** Our deep neural network learns to embed both point-cloud/natural language instruction combinations and manipulation trajectories in the same space. This allows for fast selection of a new trajectory by projecting a new environment/instruction pair and choosing its nearest-neighbor trajectory in this space.

Our Solution. Transferred trajectories become compatible across different objects when trajectories are represented 1) in the task space rather than the configuration space, and 2) relative to the object *part* in question (aligned based on its principal axis), rather than the object as a whole.

Trajectories can be represented in the task space by recording only the position and orientation of the end-effector. By doing so, we can focus on the actual interaction between the robot and the environment rather than the movement of the arm. It is very rare that the arm configuration affects the completion of the task as long as there is no collision. With the trajectory represented as a sequence of gripper position and orientation, the robot can find its arm configuration that is collision free with the environment using inverse kinematics.

However, representing the trajectory in task space is not enough to make transfers compatible. The trajectory must also be represented in a common coordinate frame regardless of the object's orientation and shape.

Thus, we align the negative z-axis along gravity and align the x-axis along the principal axis of the object *part* using PCA [23]. With this representation, even when the object part's position and orientation changes, the trajectory does not need to change. The underlying assumption is that similarly operated object parts share similar shapes leading to a similar direction in their principal axes.

IV. DEEP MULTIMODAL EMBEDDING

In this work, we use deep learning to find the most appropriate trajectory for a given point-cloud and natural language instruction. This is much more challenging than the uni-modal binary or multi-class classification/regression problems (e.g. image recognition [33]) to which deep learning has mostly been applied [6]. We could simply convert our problem into a binary classification problem, i.e. "does

this trajectory match this point-cloud/language pair?" Then, we can use a multimodal feed-forward deep network [47] to solve this problem [61].

However, this approach has several drawbacks. First, it requires evaluating the network over *every* combination of potential candidate manipulation trajectory and the given point-cloud/language pair, which is computationally expensive. Second, this method does nothing to handle noisy labels, a significant problem when dealing with crowd-sourced data as we do here. Finally, while this method is capable of producing reasonable results for our problem, we show in Section VIII-C that a more principled approach to our problem is able to improve over it.

Instead, in this work, we present a new deep architecture and algorithm which is a better fit for this problem, directly addressing the challenges inherent in multimodal data and the noisy labels obtained from crowdsourcing. We learn a joint embedding of point-cloud, language, and trajectory data into the same low dimensional space. We learn non-linear embeddings using a deep learning approach which maps raw data from these three different modalities to a joint embedding space.

We then use this space to find known trajectories which are good matches for new combinations of object parts and instructions. Compared to previous work that exhaustively runs a full network over all these combinations [61], our approach allows us to pre-embed all candidate trajectories into this common feature space. Thus, the most appropriate trajectory can be identified by embedding only a new point-cloud/language pair and then finding its nearest neighbor.

In our joint feature space, proximity between two mapped points should reflect how relevant two data-points are to each other, even if they are from completely different modalities. We train our network to bring demonstrations that would manipulate a given object according to some language instruction closer to the mapped point for that object/instruction pair, and to push away demonstrations that would not correctly manipulate the object. Trajectories which have no semantic relevance to the object are pushed much further away than trajectories that have some relevance, even if the latter would not manipulate the object according to the instruction.

Prior to learning a full joint embedding of all three modalities, we pre-train embeddings of subsets of the modalities to learn semantically meaningful embeddings for these modalities, leading to improved performance as shown in Section VIII-C.

To solve this problem of learning to manipulate novel objects and appliance as defined in equation (1), we learn two different mapping functions that map to a common space—one from a point-cloud/language pair and the other from a trajectory. More formally, we want to learn $\Phi_{\mathcal{P},\mathcal{L}}(p,l)$ and $\Phi_{\mathcal{T}}(\tau)$ which map to a joint feature space $\mathbb{R}^M$:

$$\Phi_{\mathcal{P},\mathcal{L}}(p,l) : (\mathcal{P},\mathcal{L}) \to \mathbb{R}^M$$
$$\Phi_{\mathcal{T}}(\tau) : \mathcal{T} \to \mathbb{R}^M$$

Here, we represent these mappings with a deep neural network, as shown in Figure 3.

The first, $\Phi_{\mathcal{P},\mathcal{L}}$, which maps point-clouds and trajectories, is defined as a combination of two mappings. The first of these maps to a joint point-cloud/language space $\mathbb{R}^{N_{2,pl}}$ — $\Phi_{\mathcal{P}}(p) : \mathcal{P} \to \mathbb{R}^{N_{2,pl}}$ and $\Phi_{\mathcal{L}}(l) : \mathcal{L} \to \mathbb{R}^{N_{2,pl}}$. Once each is mapped to $\mathbb{R}^{N_{2,pl}}$, this space is then mapped to the joint space shared with trajectory information: $\Phi_{\mathcal{P},\mathcal{L}}(p,l) : ((\mathcal{P},\mathcal{L}) \to \mathbb{R}^{N_{2,pl}}) \to \mathbb{R}^M$.

A. Model

We use two separate multi-layer deep neural networks, one for $\Phi_{\mathcal{P},\mathcal{L}}(p,l)$ and one for $\Phi_{\mathcal{T}}(\tau)$. Take N_p as the size of point-cloud input p, N_l as similar for natural language input l, $N_{1,p}$ and $N_{1,l}$ as the number of hidden units in the first hidden layers projected from point-cloud and natural language features, respectively, and $N_{2,pl}$ as the number of hidden units in the combined point-cloud/language layer. With W's as network weights, which are the learned parameters of our system, and $a(\cdot)$ as a rectified linear unit (ReLU) activation function [78], our model for projecting from point-cloud and language features to the shared embedding h^3 is as follows:

$$h_i^{1,p} = a\left(\sum_{j=0}^{N_p} W_{i,j}^{1,p} p_j\right)$$
$$h_i^{1,l} = a\left(\sum_{j=0}^{N_l} W_{i,j}^{1,l} l_j\right)$$
$$h_i^{2,pl} = a\left(\sum_{j=0}^{N_{1,p}} W_{i,j}^{2,p} h_j^{1,p} + \sum_{j=0}^{N_{1,l}} W_{i,j}^{2,l} h_j^{1,l}\right)$$
$$h_i^3 = a\left(\sum_{j=0}^{N_{2,pl}} W_{i,j}^{3,pl} h_j^{2,pl}\right)$$

The model for projecting from trajectory input τ is similar, except it takes input only from a single modality.

B. Inference.

Once all mappings are learned, we solve the original problem from equation (1) by choosing, from a library of prior trajectories, the trajectory that gives the highest similarity (closest in distance) to the given point-cloud p and language l in our joint embedding space $\mathbb{R}^M$. As in previous work [73], similarity is defined as $sim(a,b) = a \cdot b$, and the trajectory that maximizes the magnitude of similarity is selected:

$$\operatorname*{argmax}_{\tau \in \mathcal{T}} sim(\Phi_{\mathcal{P},\mathcal{L}}(p,l), \Phi_{\mathcal{T}}(\tau))$$

The previous approach to this problem [61] required projecting the combination of the current point-cloud and natural language instruction with *every* trajectory in the training set through the network during inference. Here, we pre-compute the representations of all training trajectories in h^3, and need only project the new point-cloud/language pair to h^3 and find its nearest-neighbor trajectory in this embedding space. As shown in Section VIII-C, this significantly improves both the runtime and accuracy of our approach and makes it much more scalable to larger training datasets like those collected with crowdsourcing platforms.

V. Learning Joint Point-cloud/Language/Trajectory Model

The main challenge of our work is to learn a model which maps three disparate modalities – point-clouds, natural language, and trajectories – to a single semantically meaningful space. We introduce a method that learns a common point-cloud/language/trajectory space such that all trajectories relevant to a given task (point-cloud/language combination) should have higher similarity to the projection of that task than task-irrelevant trajectories. Among these irrelevant trajectories, some might be less relevant than others, and thus should be pushed further away.

For example, given a door knob that needs to be grasped normal to the door surface and an instruction to rotate it clockwise, a trajectory that correctly approaches the door knob but rotates counter-clockwise should have higher similarity to the task than one which approaches the knob from a completely incorrect angle and does not execute any rotation.

For every training point-cloud/language pair (p_i, l_i), we have two sets of demonstrations: a set of trajectories $\mathcal{T}_{i,S}$ that are relevant (similar) to this task and a set of trajectories $\mathcal{T}_{i,D}$ that are irrelevant (dissimilar) as described in Sec. V-C. For each pair of (p_i, l_i), we want all projections of $\tau_j \in \mathcal{T}_{i,S}$ to have higher similarity to the projection of (p_i, l_i) than $\tau_k \in \mathcal{T}_{i,D}$. A simple approach would be to train the network to distinguish these two sets by enforcing a finite distance (safety margin) between the similarities of these two sets [72], which can be written in the form of a constraint:

$$sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau_j)) \geq 1 + sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau_k))$$

Rather than simply being able to distinguish two sets, we want to learn semantically meaningful embedding spaces from different modalities. Recalling our earlier example where one incorrect trajectory for manipulating a door knob was much closer to correct than another, it is clear that our learning algorithm should drive some of incorrect trajectories to be more dissimilar than others. The difference between the similarities of τ_j and τ_k to the projected point-cloud/language pair (p_i, l_i) should be at least the loss $\Delta(\tau_j, \tau_k)$. This can be written as a form of a constraint:

$$\forall \tau_j \in \mathcal{T}_{i,S}, \forall \tau_k \in \mathcal{T}_{i,D}$$
$$sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau_j))$$
$$\geq \Delta(\tau_j, \tau_k) + sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau_k))$$

Intuitively, this forces trajectories with higher DTW-MT distance from the ground truth to embed further than those with lower distance. Enforcing all combinations of these constraints could grow exponentially large. Instead, similar to the cutting plane method for structural support vector machines [68], we find the most violating trajectory $\tau' \in \mathcal{T}_{i,D}$ for each training pair of $(p_i, l_i, \tau_i \in \mathcal{T}_{i,S})$ at each iteration. The most violating trajectory has the highest similarity augmented with the loss scaled by a constant α:

$$\tau'_i = \arg\max_{\tau \in \mathcal{T}_{i,D}} (sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau)) + \alpha\Delta(\tau_i, \tau))$$
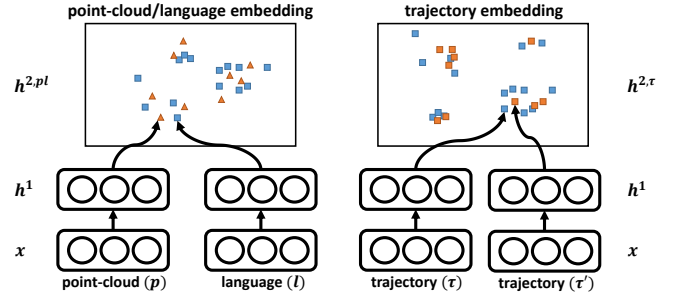

Fig. 4. **Pre-training lower layers:** Visualization of our pre-training approaches for $h^{2,pl}$ and $h^{2,\tau}$. For $h^{2,pl}$, our algorithm pushes matching point-clouds and instructions to be more similar. For $h^{2,\tau}$, our algorithm pushes trajectories with higher DTW-MT similarity to be more similar.

The cost of our deep embedding space h^3 is computed as the hinge loss of the most violating trajectory.

$$L_{h^3}(p_i, l_i, \tau_i) = |\Delta(\tau'_i, \tau_i) + sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau'_i))$$
$$- sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau_i))|_+$$

The average cost of each minibatch is back-propagated through all the layers of the deep neural network using the AdaDelta [77] algorithm.

A. Pre-training Joint Point-cloud/Language Model

One major advantage of modern deep learning methods is the use of unsupervised pre-training to initialize neural network parameters to a good starting point before the final supervised fine-tuning stage. Pre-training helps these high-dimensional networks to avoid overfitting to the training data.

Our lower layers $h^{2,pl}$ and $h^{2,\tau}$ represent features extracted exclusively from the combination of point-clouds and language, and from trajectories, respectively. Our pre-training method initializes $h^{2,pl}$ and $h^{2,\tau}$ as semantically meaningful embedding spaces similar to h^3, as shown later in Section VIII-C.

First, we pre-train the layers leading up to these layers using spare de-noising autoencoders [71, 78]. Then, our process for pre-training $h^{2,pl}$ is similar to our approach to fine-tuning a semantically meaningful embedding space for h^3 presented above, except now we find the most violating language l' while still relying on a loss over the associated optimal trajectory:

$$l' = \arg\max_{l \in \mathcal{L}}(sim(\Phi_{\mathcal{P}}(p_i), \Phi_{\mathcal{L}}(l)) + \alpha\Delta(\tau, \tau^*_i))$$

$$L_{h^{2,pl}}(p_i, l_i, \tau_i) = |\Delta(\tau_i, \tau') + sim(\Phi_{\mathcal{P}}(p_i), \Phi_{\mathcal{L}}(l'))$$
$$- sim(\Phi_{\mathcal{P}}(p_i), \Phi_{\mathcal{L}}(l_i))|_+$$

Notice that although we are training this embedding space to project from point-cloud/language data, we guide learning using trajectory information.

After the projections $\Phi_{\mathcal{P}}$ and $\Phi_{\mathcal{L}}$ are tuned, the output of these two projections are added to form the output of layer $h^{2,pl}$ in the final feed-forward network.

B. Pre-training Trajectory Model

For our task of inferring manipulation trajectories for novel objects, it is especially important that similar trajectories τ map to similar regions in the feature space defined by $h^{2,\tau}$, so that trajectory embedding $h^{2,\tau}$ itself is semantically meaningful and they can in turn be mapped to similar regions in h^3. Standard pretraining methods, such as sparse de-noising autoencoder [71, 78] would only pre-train $h^{2,\tau}$ to reconstruct individual trajectories. Instead, we employ pre-training similar to Sec. V-A, except now we pre-train for only a single modality – trajectory data.

As shown on right hand side of Fig. 4, the layer that embeds to $h^{2,\tau}$ is duplicated. These duplicated embedding layers are treated as if they were two different modalities, but all their weights are shared and updated simultaneously. For every trajectory $\tau \in \mathcal{T}_{i,S}$, we can again find the most violating $\tau' \in \mathcal{T}_{i,D}$ and the minimize a similar cost function as we do for $h^{2,pl}$.

C. Label Noise

When our data contains a significant number of noisy trajectories τ, e.g. due to crowd-sourcing (Sec. VII), not all trajectories should be trusted as equally appropriate, as will be shown in Sec. VIII.

For every pair of inputs (p_i, l_i), we have $\mathcal{T}_i = \{\tau_{i,1}, \tau_{i,2}, ..., \tau_{i,n_i}\}$, a set of trajectories submitted by the crowd for (p_i, l_i). First, the best candidate label $\tau_i^* \in \mathcal{T}_i$ for (p_i, l_i) is selected as the one with the smallest average trajectory distance to the others:

$$\tau_i^* = \operatorname*{argmin}_{\tau \in \mathcal{T}_i} \frac{1}{n_i} \sum_{j=1}^{n_i} \Delta(\tau, \tau_{i,j})$$

We assume that at least half of the crowd tried to give a reasonable demonstration. Thus a demonstration with the smallest average distance to all other demonstrations must be a good demonstration. We use the DTW-MT distance function (described later in Sec. VI) for our loss function $\Delta(\tau, \bar{\tau})$, but it could be replaced by any function that computes the loss of predicting $\bar{\tau}$ when τ is the correct demonstration.

Using the optimal demonstration and a loss function $\Delta(\tau, \bar{\tau})$ for comparing demonstrations, we find a set of trajectories $\mathcal{T}_{i,S}$ that are relevant (similar) to this task and a set of trajectories $\mathcal{T}_{i,D}$ that are irrelevant (dissimilar.) We can use thresholds (t_S, t_D) determined by the expert to generate two sets from the pool of trajectories:

$$\mathcal{T}_{i,S} = \{\tau \in \mathcal{T} | \Delta(\tau_i^*, \tau) < t_S\}$$

$$\mathcal{T}_{i,D} = \{\tau \in \mathcal{T} | \Delta(\tau_i^*, \tau) > t_D\}$$

This method allows our model to be robust against noisy labels and also serves as a method of data augmentation by also considering demonstrations given for other tasks in both sets of $\mathcal{T}_{i,S}$ and $\mathcal{T}_{i,D}$.

VI. Loss Function for Manipulation Trajectory

For both learning and evaluation, we need a function which accurately represents distance between two trajectories. Prior metrics for trajectories consider only their translations (e.g. [31]) and not their rotations *and* gripper status. We propose a new measure, which uses dynamic time warping, for evaluating manipulation trajectories. This measure non-linearly warps two trajectories of arbitrary lengths to produce a matching, then computes cumulative distance as the sum of cost of all matched waypoints. The strength of this measure is that weak ordering is maintained among matched waypoints and that every waypoint contributes to the cumulative distance.

For two trajectories of arbitrary lengths, $\tau_A = \{\tau_A^{(i)}\}_{i=1}^{m_A}$ and $\tau_B = \{\tau_B^{(i)}\}_{i=1}^{m_B}$, we define a matrix $D \in \mathbb{R}^{m_A \times m_B}$, where $D(i, j)$ is the cumulative distance of an optimally-warped matching between trajectories up to index i and j, respectively, of each trajectory. The first column and the first row of D is initialized as:

$$D(i, 1) = \sum_{k=1}^{i} c(\tau_A^{(k)}, \tau_B^{(1)}) \quad \forall i \in [1, m_A]$$

$$D(1, j) = \sum_{k=1}^{j} c(\tau_A^{(1)}, \tau_B^{(k)}) \quad \forall j \in [1, m_B]$$

where c is a local cost function between two waypoints (discussed later). The rest of D is completed using dynamic programming:

$$D(i, j) = c(\tau_A^{(i)}, \tau_B^{(j)}) \\ + \min\{D(i-1, j-1), D(i-1, j), D(i, j-1)\}$$

Given the constraint that $\tau_A^{(1)}$ is matched to $\tau_B^{(1)}$, the formulation ensures that every waypoint contributes to the final cumulative distance $D(m_A, m_B)$. Also, given a matched pair $(\tau_A^{(i)}, \tau_B^{(j)})$, no waypoint preceding $\tau_A^{(i)}$ is matched to a waypoint succeeding $\tau_B^{(j)}$, encoding weak ordering.

The pairwise cost function c between matched waypoints $\tau_A^{(i)}$ and $\tau_B^{(j)}$ is defined:

$$c(\tau_A^{(i)}, \tau_B^{(j)}; \alpha_T, \alpha_R, \beta, \gamma) = w(\tau_A^{(i)}; \gamma) w(\tau_B^{(j)}; \gamma) \\ \left(\frac{d_T(\tau_A^{(i)}, \tau_B^{(j)})}{\alpha_T} + \frac{d_R(\tau_A^{(i)}, \tau_B^{(j)})}{\alpha_R} \right) \left(1 + \beta d_G(\tau_A^{(i)}, \tau_B^{(j)}) \right)$$

where

$$d_T(\tau_A^{(i)}, \tau_B^{(j)}) = ||(t_x, t_y, t_z)_A^{(i)} - (t_x, t_y, t_z)_B^{(j)}||_2$$
$$d_R(\tau_A^{(i)}, \tau_B^{(j)}) = \text{angle difference between } \tau_A^{(i)} \text{ and } \tau_B^{(j)}$$
$$d_G(\tau_A^{(i)}, \tau_B^{(j)}) = \mathbb{1}(g_A^{(i)} = g_B^{(j)})$$
$$w(\tau^{(i)}; \gamma) = exp(-\gamma \cdot ||\tau^{(i)}||_2)$$

The parameters α, β are for scaling translation and rotation errors, and gripper status errors, respectively. γ weighs the importance of a waypoint based on its distance to the object part. [2] Finally, as trajectories vary in length, we normalize $D(m_A, m_B)$ by the number of waypoint pairs that contribute

---

[2] In this work, we assign $\alpha_T, \alpha_R, \beta, \gamma$ values of 0.0075 meters, 3.75°, 1 and 4 respectively.
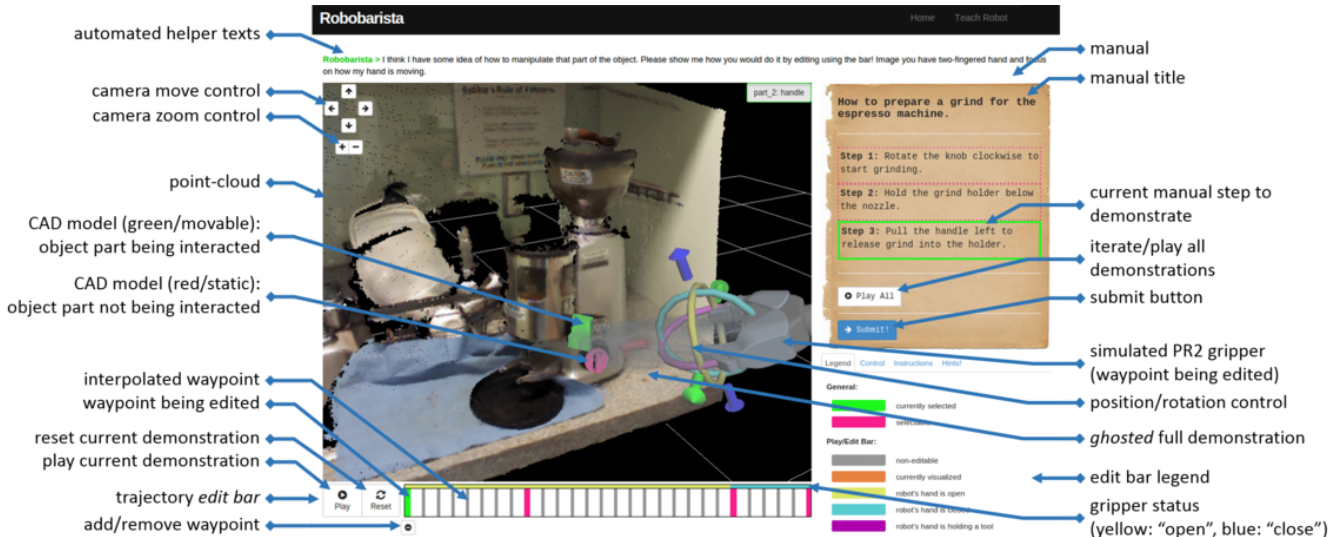
Fig. 5. **Screen-shot of Robobarista,** the crowd-sourcing platform running on Chrome browser. We have built Robobarista platform for collecting a large number of crowd demonstrations for teaching the robot.

to the cumulative sum, $|D(m_A, m_B)|_{path^*}$ (i.e. the length of the optimal warping path), giving the final form:

$$distance(\tau_A, \tau_B) = \frac{D(m_A, m_B)}{|D(m_A, m_B)|_{path^*}}$$

This distance function is used for noise-handling in our model and as the final evaluation metric.

VII. ROBOBARISTA: CROWD-SOURCING PLATFORM

In order to collect a large number of manipulation demonstrations from the crowd, we built a crowd-sourcing web platform that we call Robobarista (see Fig. 5). It provides a virtual environment where non-expert users can teach robots via a web browser, without expert guidance or physical presence with a robot and a target object.

The system simulates a situation where the user encounters a previously unseen target object and a natural language instruction manual for its manipulation. Within the web browser, users are shown a point-cloud in the 3-D viewer on the left and a *manual* on the right. A manual may involve several instructions, such as "Push down and pull the handle to open the door". The user's goal is to demonstrate how to manipulate the object in the scene for each instruction.

The user starts by selecting one of the instructions on the right to demonstrate (Fig. 5). Once selected, the target object part is highlighted and the trajectory *edit bar* appears below the 3-D viewer. Using the *edit bar*, which works like a video editor, the user can playback and edit the demonstration. The trajectory representation, as a set of waypoints (Sec. III-A), is directly shown on the *edit bar*. The bar shows not only the set of waypoints (red/green) but also the interpolated waypoints (gray). The user can click the 'play' button or hover the cursor over the edit bar to examine the current demonstration. The blurred trail of the current trajectory (*ghosted*) demonstration is also shown in the 3-D viewer to show its full expected path.

Generating a full trajectory from scratch can be difficult for non-experts. Thus, similar to Forbes et al. [17], we provide a trajectory that the system has already seen for another object as the initial starting trajectory to edit.[3]

In order to simulate a realistic experience of manipulation, instead of simply showing a static point-cloud, we have overlaid CAD models for parts such as 'handle' so that functional parts actually move as the user tries to manipulate the object.

A demonstration can be edited by: 1) modifying the position/orientation of a waypoint, 2) adding/removing a waypoint, and 3) opening/closing the gripper. Once a waypoint is selected, the PR2 gripper is shown with six directional arrows and three rings, used to modify the gripper's position and orientation, respectively. To add extra waypoints, the user can hover the cursor over an interpolated (gray) waypoint on the *edit bar* and click the plus(+) button. To remove an existing waypoint, the user can hover over it on the *edit bar* and click minus(-) to remove. As modification occurs, the edit bar and ghosted demonstration are updated with a new interpolation. Finally, for editing the status (open/close) of the gripper, the user can simply click on the gripper.

For broader accessibility, all functionality of Robobarista, including 3-D viewer, is built using Javascript and WebGL. We have made the platform available online (http://robobarista.cs.cornell.edu)

VIII. EXPERIMENTS

A. Robobarista Dataset

In order to test our model, we have collected a dataset of 116 point-clouds of objects with 249 object parts (examples shown in Figure 6). Objects range from kitchen appliances such as stoves and rice cookers to bathroom hardware such as sinks and toilets. Figure 14 shows a sample of 70 such

---

[3]We have made sure that it does not initialize with trajectories from other folds to keep *5-fold cross-validation* in experiment section valid.
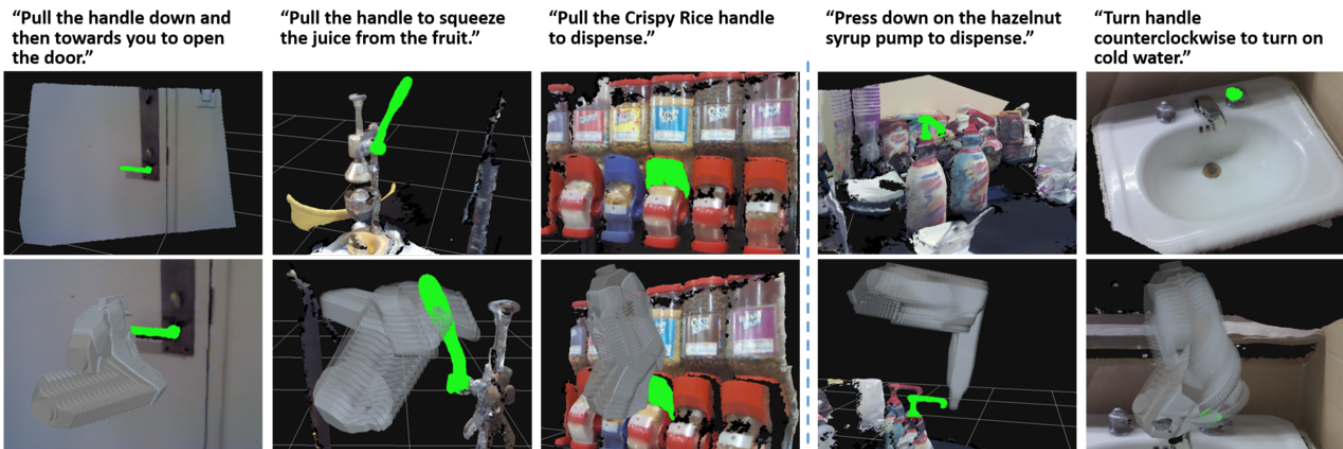
Fig. 6. **Examples from our dataset,** each of which consists of a natural language instruction (top), an object part in point-cloud representation (highlighted), and a manipulation trajectory (below) collected via Robobarista. Objects range from kitchen appliances such as stove and rice cooker to urinals and sinks in restrooms. As our trajectories are collected from non-experts, they vary in quality from being likely to complete the manipulation task successfully (left of dashed line) to being unlikely to do so successfully (right of dashed line).

objects. There are also a total of 250 natural language instructions (in 155 manuals).[4] Using the crowd-sourcing platform Robobarista, we collected 1225 trajectories for these objects from 71 non-expert users on the Amazon Mechanical Turk. After a user is shown a 20-second instructional video, the user first completes a 2-minute tutorial task. At each session, the user was asked to complete 10 assignments where each consists of an object and a manual to be followed.

For each object, we took raw RGB-D images with the Microsoft Kinect sensor and stitched them using Kinect Fusion [26] to form a denser point-cloud in order to incorporate different viewpoints of objects. Objects range from kitchen appliances such as 'stove', 'toaster', and 'rice cooker' to 'urinal', 'soap dispenser', and 'sink' in restrooms. The dataset is made available at `http://robobarista.cs.cornell.edu`

B. Baselines

We compared our model against many baselines:

1) *Random Transfers (chance)*: Trajectories are selected at random from the set of trajectories in the training set.

2) *Object Part Classifier*: To test our hypothesis that classifying object parts as an intermediate step does not guarantee successful transfers, we trained an object part classifier using multiclass SVM [67] on point-cloud features $\phi(p)$ including local shape features [32], histogram of curvatures [52], and distribution of points. Using this classifier, we first classify the target object part p into an object part category (e.g. 'handle', 'knob'), then use the same feature space to find its nearest neighbor p' of the same class from the training set. Then the trajectory τ' of p' is transferred to p.

3) *Structured support vector machine (SSVM)*: We used SSVM to learn a discriminant scoring function $\mathcal{F} : \mathcal{P} \times \mathcal{L} \times \mathcal{T} \to \mathbb{R}$. At test time, for target point-cloud/language

---

[4]Although not necessary for training our model, we also collected trajectories from the expert for evaluation purposes.

pair (p, l), we output the trajectory τ from the training set that maximizes $\mathcal{F}$. To train SSVM, we use a joint feature mapping $\phi(p, l, \tau) = [\phi(\tau); \phi(p, \tau); \phi(l, \tau)]$. $\phi(\tau)$ applies Isomap [63] to interpolated τ for non-linear dimensionality reduction. $\phi(p, \tau)$ captures the overall shape when trajectory τ is overlaid over point-cloud p by jointly representing them in a voxel-based cube similar to Sec. III-B, with each voxel holding count of occupancy by p or τ. Isomap is applied to this representation to get the final $\phi(p, \tau)$. Finally, $\phi(l, \tau)$ is the tensor product of the language features and trajectory features: $\phi(l, \tau) = \phi(l) \otimes \phi(\tau)$. We used our loss function (Sec. VI) to train SSVM and used the cutting plane method to solve the SSVM optimization problem [28].

4) *Latent Structured SVM (LSSVM) + kinematic structure*: The way in which an object is manipulated largely depends on its internal structure – whether it has a 'revolute', 'prismatic', or 'fixed' joint. Instead of explicitly trying to learn this structure, we encoded this internal structure as latent variable $z \in \mathcal{Z}$, composed of joint type, center of joint, and axis of joint [58]. We used Latent SSVM [76] to train with z, learning the discriminant function $\mathcal{F} : \mathcal{P} \times \mathcal{L} \times \mathcal{T} \times \mathcal{Z} \to \mathbb{R}$. The model was trained with feature mapping $\phi(p, l, \tau, z) = [\phi(\tau); \phi(p, \tau); \phi(l, \tau); \phi(l, z); \phi(p, \tau, z)]$, which includes additional features that involve z. $\phi(l, z)$ captures the relation between l, a bag-of-words representation of language, and bag-of-joint-types encoded by z (vector of length 3 indicating existence of each joint type) by computing the tensor product $\phi(l) \otimes \phi(z)$, then reshaping the product into a vector. $\phi(p, \tau, z)$ captures how well the portion of τ that actually interacts with p abides by the internal structure h. $\phi(p, \tau, z)$ is a concatenation of three types of features, one for each joint type. For 'revolute' type joints, it includes deviation of trajectory from plane of rotation defined by z, the maximum angular rotation while maintaining pre-defined proximity to the plane of rotation, and the average cosine similarity between rotation axis of τ

TABLE I

RESULTS ON OUR DATASET WITH *5-fold cross-validation*. ROWS LIST MODELS WE TESTED INCLUDING OUR MODEL AND BASELINES. COLUMNS SHOW DIFFERENT METRICS USED TO EVALUATE THE MODELS.

Models	per manual	per instruction	
	DTW-MT	DTW-MT	Accuracy (%)
Chance	28.0 (±0.8)	27.8 (±0.6)	11.2 (±1.0)
Object Part Classifier	-	22.9 (±2.2)	23.3 (±5.1)
Structured SVM	21.0 (±1.6)	21.4 (±1.6)	26.9 (±2.6)
Latent SSVM + Kinematic [58]	17.4 (±0.9)	17.5 (±1.6)	40.8 (±2.5)
Task similarity + Random	14.4 (±1.5)	13.5 (±1.4)	49.4 (±3.9)
Task Similarity + Weights [17]	13.3 (±1.2)	12.5 (±1.2)	53.7 (±5.8)
Deep Network with Noise-handling without Embedding	13.7 (±1.6)	13.3 (±1.6)	51.9 (±7.9)
Deep Multimodal Network without Embedding	14.0 (±2.3)	13.7 (±2.1)	49.7 (±10.0)
Deep Multimodal Network with Noise-handling without Embedding [61]	13.0 (±1.3)	12.2 (±1.1)	60.0 (±5.1)
LMNN-like Cost Function [72]	15.4 (±1.8)	14.7 (±1.6)	55.5 (±5.3)
Our Model without Any Pretraining	13.2 (±1.4)	12.4 (±1.0)	54.2 (±6.0)
Our Model with SDA	11.5 (±0.6)	11.1 (±0.6)	62.6 (±5.8)
Our Model without Noise Handling	12.6 (±1.3)	12.1 (±1.1)	53.8 (±8.0)
Our Model with Experts	12.3 (±0.5)	11.8 (±0.9)	56.5 (±4.5)
Our Model - Deep Multimodal Embedding	**11.0** (±0.8)	**10.5** (±0.7)	**65.1** (±4.9)

and axis defined by z. For 'prismatic' joints, it includes the average cosine similarity between the extension axis and the displacement vector between waypoints. Finally, for 'fixed' joints, it includes whether the *uninteracting* part of τ has collision with the background p since it is important to approach the object from correct angle.

5) *Task-Similarity Transfers + random*: We compute the pairwise similarities between the test case (p_{test}, l_{test}) and each training example (p_{train}, l_{train}), then transfer a trajectory τ associated with the training example of highest similarity. Pairwise similarity is defined as a convex combination of the cosine similarity in bag-of-words representations of language and the average mutual point-wise distance of two point-clouds after a fixed number of iterations of the ICP [7] algorithm. If there are multiple trajectories associated with (p_{train}, l_{train}) of highest similarity, the trajectory for transfer is selected randomly.

6) *Task-similarity Transfers + weighting*: The previous method is problematic when non-expert demonstrations for a single task (p_{train}, l_{train}) vary in quality. Forbes et al. [17] introduces a score function for weighting demonstrations based on weighted distance to the "seed" (expert) demonstration. Adapting to our scenario of not having any expert demonstrations, we select τ that has the lowest average distance from all other demonstrations for the same task, with each distance measured with our loss function (Sec. VI.) This is similar to our noise handling approach in Sec. V-C.

7) *Deep Network without Embedding*: We train a deep neural network to learn a similar scoring function $\mathcal{F}: \mathcal{P} \times \mathcal{L} \times \mathcal{T} \to \mathbb{R}$ to that learned for SSVM above. This model discriminatively projects the combination of point-cloud, language, and trajectory features to a score which represents how well the trajectory matches that point-cloud/language combination. Note that this is much less efficient than our joint embedding approach, as it must consider all combinations of a new point-cloud/language pair and every training trajectory to perform inference, as opposed to our model which need only project this new pair to our joint embedding space. This deep
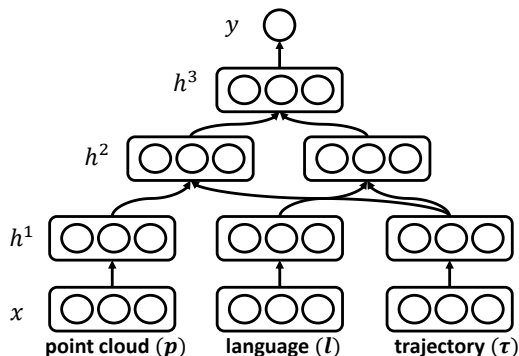

Fig. 7. **Deep Multimodal Network without Embedding** baseline model takes the input x of three different modalities (point-cloud, language, and trajectory) and outputs y, whether it is a good match or bad match. It first learns features separately (h^1) for each modality and then learns the relation (h^2) between input and output of the original structured problem. Finally, last hidden layer h^3 learns relations of all these modalities.

learning model concatenates all the input of three modalities and learns three hidden layers before the final layer.

8) *Deep Multimodal Network without Embedding*: The same approach as 'Deep Network without Embedding' with layers per each modality before concatenating as shown in Figure 7. More details about the model can be found in [61].

9) *LMNN [72]-like cost function:* For all top layer fine-tuning and lower layer pre-training, we define the cost function without loss augmentation. Similar to LMNN [72], we give a finite margin between similarities. For example, as cost function for h^3:

$$L_{h^3}(p_i, l_i, \tau_i) = |1 + sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau')) \\ - sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau_i))|_+$$

10) *Our Model without Pretraining:* Our full model finetuned without any pre-training of lower layers – all parameters are randomly initialized.

11) *Our Model with SDA:* Our full model without pre-training $h^{2,pl}$ and $h^{2,\tau}$ as defined in Section V-A and

Successful Transfers

"Pull the Colossal Crunch handle to dispense." → "Pull down on the right handle to dispense the coffee."

"Turn the switch clockwise to switch on the power supply" → "Rotate the knob clockwise to turn slow cooker on."

Unsuccessful Transfer

"Turn the handle counterclockwise to unlock the cooker." → "Turn the knob counterclockwise to decrease the temperature"
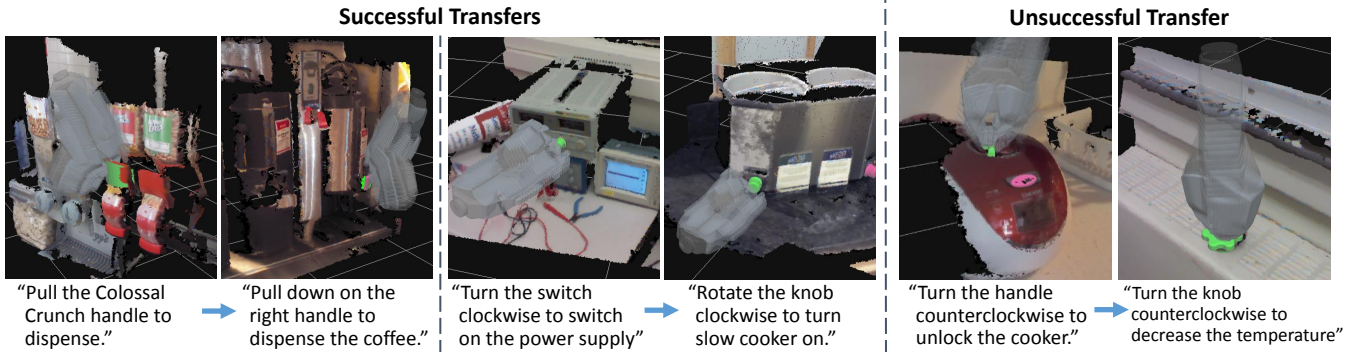
Fig. 8. **Examples of successful and unsuccessful transfers** of manipulation trajectory from left to right using our model. In first two examples, though the robot has never seen the 'coffee dispenser' and 'slow cooker' before, the robot has correctly identified that the trajectories of 'cereal dispenser' and 'DC power supply', respectively, can be used to manipulate them.

Section V-B. Instead, we pre-train each layer as stacked denoising autoencoders [71, 78].

12) *Our Model without Noise Handling:* Our model is trained without noise handling as presented in Section V-C. All of the trajectories collected from the crowd are trusted as a ground-truth labels.

13) *Our Model with Experts:* Our model is trained only using trajectory demonstrations from an expert which were collected for evaluation purposes.

14) *Our Full Model - Deep Multimodal Embedding:* Our full model as described in this paper with network size of $h^{1,p}$, $h^{1,l}$, $h^{1,\tau}$, $h^{2,pl}$, $h^{2,\tau}$, and h^3 respectively having a layer with $150, 175, 100, 100, 75$, and 50 nodes.

C. Results and Discussions

We evaluated all models on our dataset using *5-fold cross-validation* and the results are in Table I. All models which required hyper-parameter tuning used 10% of the training data as the validation set.

Rows list the models we tested including our model and baselines. Each column shows one of three evaluation metrics. The first two use dynamic time warping for manipulation trajectory (DTW-MT) from Sec. VI. The first column shows averaged DTW-MT for each instruction manual consisting of one or more language instructions. The second column shows averaged DTW-MT for every test pair (p, l).

As DTW-MT values are not intuitive, we also include a measure of "accuracy," which shows the percentage of transferred trajectories with DTW-MT value less than 10. Through expert surveys, we found that when DTW-MT of manipulation trajectory is less than 10, the robot came up with a reasonable trajectory and will very likely be able to accomplish the given task. Additionally, Fig. 11 shows accuracies obtained by varying the threshold on the DTW-MT measure.

Can manipulation trajectories be transferred from completely different objects? Our full model gave 65.1% accuracy (Table I), outperforming every other baseline approach tested.

Fig. 8 shows two examples of successful transfers and one unsuccessful transfer by our model. In the first example,

the trajectory for pulling down on a cereal dispenser is transferred to a coffee dispenser. Because our approach to trajectory representation is based on the principal axis (Sec. III-C), even though the cereal and coffee dispenser handles are located and oriented differently, the transfer is a success. The second example shows a successful transfer from a DC power supply to a slow cooker, which have "knobs" of similar shape. The transfer was successful despite the difference in instructions ("Turn the switch.." and "Rotate the knob..") and object type. This highlights the advantages of our end-to-end approach over relying on semantic classes for parts and actions.

The last example in Fig. 8 shows a potentially unsuccessful transfer. Despite the similarity in two instructions and similarity in required counterclockwise motions, the transferred motion might not be successful. While the knob on radiator must be grasped in the middle, the rice cooker has a handle that extends sideways, requiring it to be grasped off-center. For clarity of visualization in figures, we have overlaid CAD models over some noisy point-clouds. Many of the object parts were too small and/or too glossy for the Kinect sensor. We believe that a better 3-D sensor would allow for more accurate transfers. On the other hand, it is interesting to note that the transfer in opposite direction from the radiator knob to the rice cooker handle may have yielded a correct manipulation.

Can we crowd-source the teaching of manipulation trajectories? When we trained our full model with expert demonstrations, which were collected for evaluation purposes, it performed at 56.5% compared to 65.1% by our model trained with crowd-sourced data. Even though non-expert demonstrations can carry significant noise, as shown in last two examples of Fig. 6, our noise-handling approach allowed our model to take advantage of the larger, less accurate crowd-sourced dataset. Note that all of our crowd users are true non-expert users from Amazon Mechanical Turk.

Is segmentation required for the system? Even with the state-of-the-art techniques [16, 33], detection of 'manipulable' object parts such as 'handles' and 'levers' in a point-

Fig. 10. **Examples of transferred trajectories** being executed on PR2. On the left, PR2 is able to rotate the 'knob' to turn the lamp on. In the third snapshot, using two transferred trajectories, PR2 is able to hold the cup below the 'nozzle' and press the 'lever' of 'coffee dispenser'. In the last example, PR2 is frothing milk by pulling down on the lever, and is able to prepare a cup of latte with many transferred trajectories.

Fig. 9. **Comparisons of transfers** between our model and the baseline (deep multimodal network without embedding [61]). In these three examples, our model successfully finds correct manipulation trajectory from these objects while the other one does not. Given the lever of the toaster, our algorithm finds similarly slanted part from the rice cooker while the other model finds completely irrelevant trajectory. For the opening action of waffle maker, trajectory for paper cutter is correctly identified while the other model transfers from a handle that has incompatible motion.

cloud is by itself a challenging problem [35]. Thus, we rely on human experts to pre-label parts of the object to be manipulated. The point-cloud of the scene is over-segmented into thousands of supervoxels, from which the expert chooses the part of the object to be manipulated. Even with expert input, such segmented point-clouds are still extremely noisy because of sensor failures, e.g. on glossy surfaces.

Is intermediate object part labeling necessary? A multiclass SVM trained on object part labels was able to obtain over 70% recognition accuracy in classifying five major classes of object parts ('button', 'knob', 'handle', 'nozzle', 'lever'.) However, the *Object Part Classifier* baseline, based on this classification, performed at only 23.3% accuracy for actual trajectory transfer, outperforming chance by merely 12.1%, and significantly underperforming our model's result of 65.1%. This shows that object part labels alone are not sufficient to enable manipulation motion transfer, while our model, which makes use of richer information, does a much better job.

Can features be hand-coded? What does learned deep embedding space represent? Even though we carefully designed state-of-the-art task-specific features for the SSVM and LSSVM models, these models only gave at most 40.8% accuracy. The *task similarity* method gave a better result of 53.7%, but it requires access to *all* of the raw training data (point-clouds, language, and trajectories) at test time, which leads to heavy computation at test time and requires a large amount of storage as the size of training data increases. Our approach, by contrast, requires only the trajectory data, and a low-dimensional representation of the point-cloud and language data, which is much less expensive to store than the raw data.

This shows that it is extremely difficult to find a good set of features which properly combines these three modalities. Our multimodal embedding model does not require hand-designing such features, instead learning a joint embedding space as shown by our visualization of the top layer h^3 in Figure 12. This visualization is created by projecting all training data (point-cloud/language pairs and trajectories) of one of the cross-validation folds to h^3, then embedding them to 2-dimensional space using t-SNE [69]. Although previous work [61] was able to visualize several nodes in the top layer, most were difficult to interpret. With our model, we can embed all our data and visualize all the layers (see Figs. 12 and 13).

One interesting result is that our system was able to naturally learn that "nozzle" and "spout" are effectively synonyms for purposes of manipulation. It clustered these together in the lower-right of Fig. 12 based solely on the fact that both are associated with similar point-cloud shapes and manipulation trajectories. At the same time, it also identified one exception, a small cluster of "nozzles" in the center of Fig. 12 which require different manipulation motions.
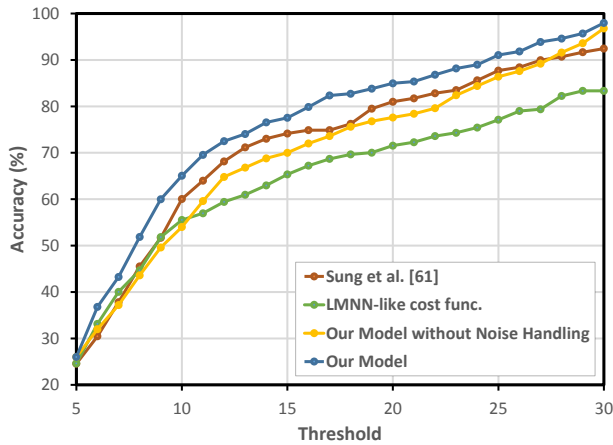
Fig. 11. **Thresholding Accuracy:** Accuracy-threshold graph showing results of varying thresholds on DTW-MT scores. Our algorithm consistently outperforms the previous approach [61] and an LMNN-like cost function [72].

In addition to the aforementioned cluster in the bottom-right of Fig. 12, we see several other logical clusters. Importantly, we can see that our embedding maps vertical and horizontal rotation operations to very different regions of the space – roughly 12 o'clock and 8 o'clock in Fig. 12, respectively. Even though these have nearly identical language instructions, our algorithm learns to map them differently based on their point-clouds, mapping nearby the appropriate manipulation trajectories.

Should cost function be loss-augmented? When we changed the cost function for pre-training h^2 and fine-tuning h^3 to use a constant margin of 1 between relevant $\mathcal{T}_{i,S}$ and irrelevant $\mathcal{T}_{i,D}$ demonstrations [72], performance drops to 55.5%. This loss-augmentation is also visible in our embedding space. Notice the purple cluster around the 6 o'clock region of Fig. 12, and the lower part of the cluster in the 5 o'clock region. The purple cluster represents tasks and demonstrations related to pushing a bar (often found on soda fountains), and the lower part of the red cluster represents the task of holding a cup below the nozzle. Although the motion required for one task would not be replaceable by the other, the motions and shapes are very similar, especially compared to most other motions e.g. turning a horizontal knob.

Is pre-embedding important? As seen in Table I, without any pre-training our model gives an accuracy of only 54.2%. Pre-training the lower layers with the conventional stacked de-noising auto-encoder (SDA) algorithm [71, 78] increases performance to 62.6%, still significantly underperforming our pre-training algorithm, which gives 65.1%. This shows that our metric embedding pre-training approach provides a better initialization for an embedding space than SDA.

Fig. 13 shows the joint point-cloud/language embedding $h^{2,pl}$ after the network is initialized using our pre-training algorithm and then fine-tuned using our cost function for h^3. While this space is not as clearly clustered as h^3 shown in Fig. 12, we note that point-clouds tend to appear in the more general center of the space, while natural language instructions appear around the more-specific edges. This

makes sense because one point-cloud might afford many possible actions, while language instructions are much more specific.

Does embedding improve efficiency? The previous model [61] had $749,638$ parameters to be learned, while our model has only $418,975$ (and still gives better performance.)

The previous model had to compute joint point-cloud/language/trajectory features for all combinations of the current point-cloud/language pair with *each* candidate trajectory (i.e. all trajectories in the training set) to infer an optimal trajectory. This is inefficient and does not scale well with the number of training datapoints. However, our model pre-computes the projection of all trajectories into h^3. Inference in our model then requires only projecting the new point-cloud/language combination to h^3 once and finding the trajectory with maximal similarity in this embedding.

In practice, this results in a significant improvement in efficiency, decreasing the average time to infer a trajectory from 2.3206ms to 0.0135ms, a speed-up of about 171x. Time was measured on the same hardware, with a GPU (GeForce GTX Titan X), using the Theano library [5]. We measured inference times 10000 times for first test fold, which has a pool of 962 trajectories. Time to preprocess the data and time to load into GPU memory was not included in this measurement. We note that the only part of our algorithm's runtime which scales up with the amount of training data is the nearest-neighbor computation, for which there exist many efficient algorithms [45]. Thus, our algorithm could be scaled to much larger datasets, allowing it to handle a wider variety of tasks, environments, and objects.

D. Robotic Validation

To validate the concept of part-based manipulation trajectory transfer in a real-world setting, we tested our algorithm on our PR2 robot. To ensure real transfers, we tested with four objects the algorithm had never seen before – a coffee dispenser, coffee grinder, lamp, and espresso machine.

The PR2 robot has two 7DoF arms, an omni-directional base, and many sensors including a Microsoft Kinect, stereo cameras, and a tilting laser scanner. For these experiments, a point-cloud is acquired from the head mounted Kinect sensor and each motion is executed on the specified arm using a Cartesian end-effector stiffness controller [9] in ROS [51].

For each object, the robot is presented with a segmented point-cloud along with a natural language text manual, with each step in the manual associated with a segmented part in the point-cloud. Once our algorithm outputs a trajectory (transferred from a completely different object), we find the manipulation frame for the part's point-cloud by using its principal axis (Sec. III-B). Then, the transferred trajectory can be executed relative to the part using this coordinate frame, without any modification to the trajectory.

The chosen manipulation trajectory, defined as a set of waypoints, is converted to a smooth and densely interpolated trajectory (Sec. III-A.) The robot first computes and execute a collision-free motion to the starting point of the manipulation trajectory. Then, starting from this first waypoint, the
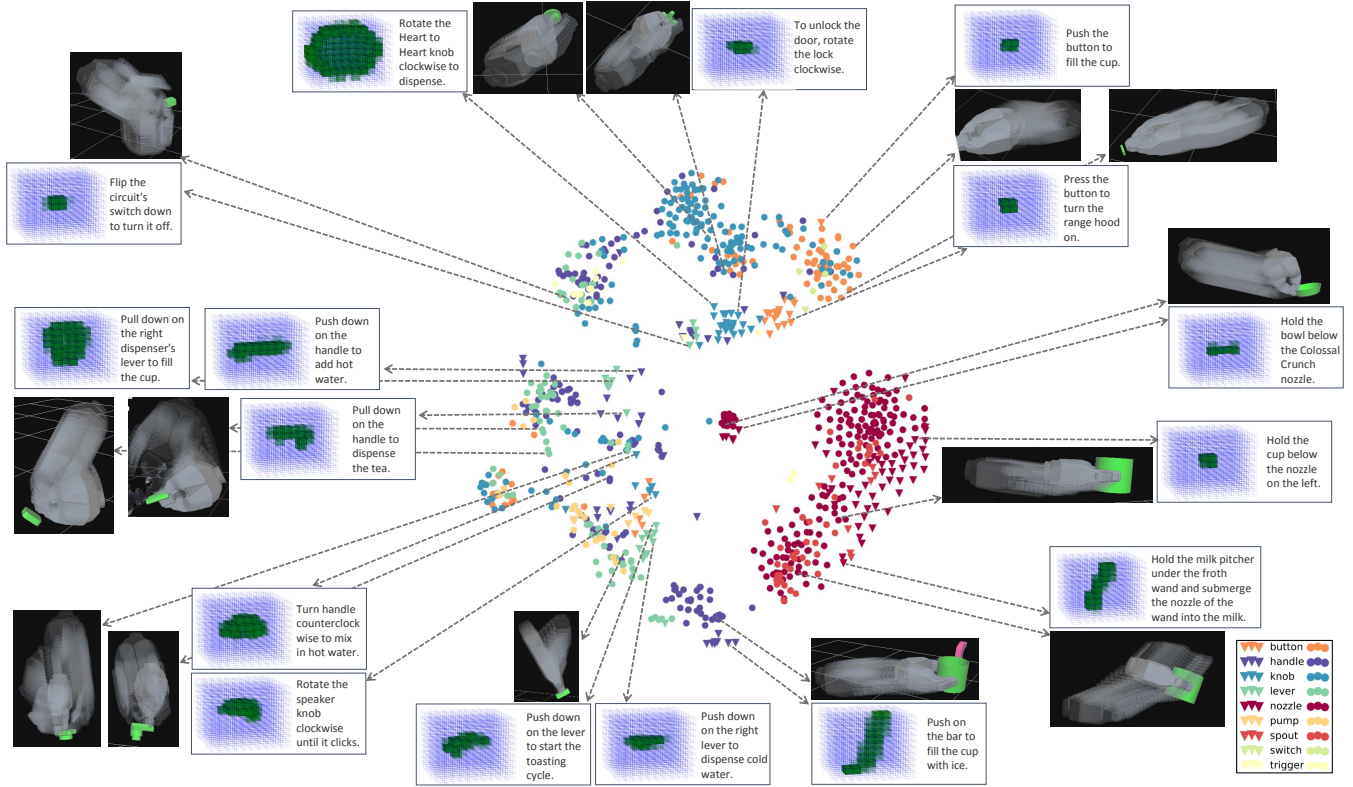
Fig. 12. **Learned Deep Point-cloud/Language/Trajectory Embedding Space:** Joint embedding space h^3 after the network is fully fine-tuned, visualized in 2d using t-SNE [69] . Inverted triangles represent projected point-cloud/language pairs, circles represent projected trajectories. The occupancy grid representation of object part point-clouds is shown in green in blue grids. Among the two occupancy grids (Sec. III-B), we selected the one that is more visually parsable for each object. The legend at the bottom right shows classifications of object parts by an expert, collected for the purpose of building a baseline. As shown by result of this baseline (object part classifier in Table I), these labels do not necessarily correlate well with the actual manipulation motion. Thus, full separation according to the labels defined in the legend is not optimal and will not occur in this figure or Fig. 13. These figures are best viewed in color.

interpolated trajectory is executed. For these experiments, we placed the robot in reach of the object, but one could also find a location using a motion planner that would make all waypoints of the manipulation trajectory reachable.

Some of the examples of successful execution on a PR2 robot are shown in Fig. 10 and in video at the project website: `http://robobarista.cs.cornell.edu/`. For example, a manipulation trajectory from the task of "turning on a light switch" is transferred to the task of "flipping on a switch to start extracting espresso", and a trajectory for turning on DC power supply (by rotating clockwise) is transferred to turning on the floor lamp. These demonstrations shows that part-based transfer of manipulation trajectories is feasible without any modification to the source trajectories by carefully choosing their representation and coordinate frames (Sec. III-C).

IX. Conclusion and Future Work

In this work, we introduce a novel approach to predicting manipulation trajectories via part based transfer, which allows robots to successfully manipulate even objects they have never seen before. We formulate this as a structured-output problem and approach the problem of inferring manipulation trajectories for novel objects by jointly embedding point-cloud, natural language, and trajectory data into a common space using a deep neural network. We introduce a method for learning a common representation of multimodal data using a new loss-augmented cost function, which learns a semantically meaningful embedding from data. We also introduce a method for pre-training the network's lower layers, learning embeddings for subsets of modalities, and show that it outperforms standard pre-training algorithms. Learning such an embedding space allows efficient inference by comparing the embedding of a new point-cloud/language pair against pre-embedded demonstrations. We introduce our crowd-sourcing platform, Robobarista, which allows non-expert users to easily give manipulation demonstrations over the web. This enables us to collect a large-scale dataset of 249 object parts with 1225 crowd-sourced demonstrations, on which our algorithm outperforms all other methods tried. We also verify on our robot that even manipulation trajectories transferred from completely different objects can be used to successfully manipulate novel objects the robot has never seen before.

While our model is able to give correct manipulation trajectories for most of the objects we tested it on, outperform-
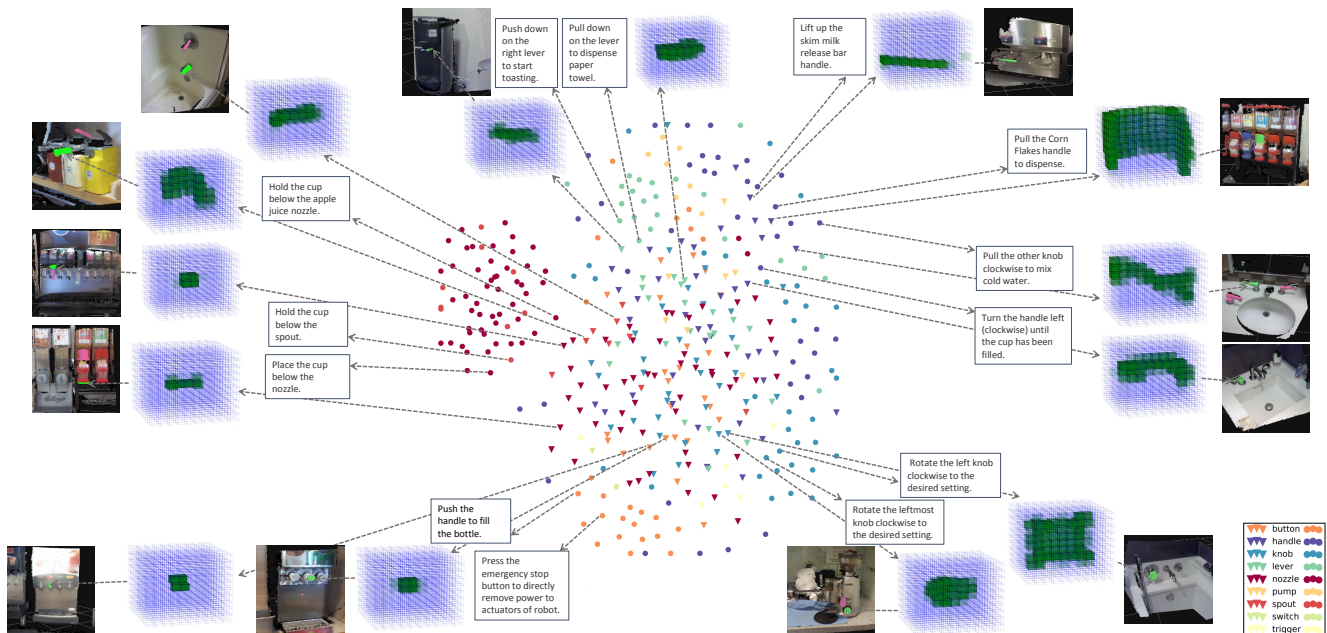
Fig. 13. **Learned Point-cloud/Language Space:** Visualization of the point-cloud/language layer $h^{2,lp}$ in 2d using t-SNE [69] after the network is fully fine-tuned. Inverted triangles represent projected point-clouds and circles represent projected instructions. A subset of the embedded points are randomly selected for visualization. Since 3D point-clouds of object parts are hard to visualize, we also include a snapshot of a point-cloud showing the whole object. Notice correlations in the motion required to manipulate the object or follow the instruction among nearby point-clouds and natural language.

ing all other approaches tried, open-loop execution of a pose trajectory may not be enough to correctly manipulate some objects. For such objects, correctly executing a transferred manipulation trajectory may require incorporating visual and/or force feedback [74, 70] in order for the execution to adapt exactly to the object. For example, some dials or buttons have to be rotated or pushed until they click, and each might require a different amount of displacement to accomplish this task. For such tasks, the robot would have to use this feedback to adapt its trajectory in an online fashion.

Our current model also only takes into account the object part and desired action in question. For some objects, a correct trajectory according to these might still collide with other parts of the environment. Once again, solving this problem would require adapting the manipulation trajectory after it's selected, and is an interesting direction for future work.

ACKNOWLEDGMENT

REFERENCES

[1] P. Abbeel, A. Coates, and A. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *International Journal of Robotics Research*, 2010.

[2] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine learning*, 1991.

[3] B. Alexander, K. Hsiao, C. Jenkins, B. Suay, and R. Toris. Robot web tools [ros topics]. *Robotics & Automation Magazine, IEEE*, 19(4):20–23, 2012.

[4] B. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *RAS*, 2009.

[5] F. Bastien, P. Lamblin, R. Pascanu, et al. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[6] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1798–1828, 2013.

[7] P. J. Besl and N. D. McKay. Method for registration of 3-d shapes. In *Robotics-DL tentative*, pages 586–606. International Society for Optics and Photonics, 1992.

[8] M. Blaschko and C. Lampert. Learning to localize objects with structured output regression. In *ECCV*. 2008.

[9] M. Bollini, J. Barry, and D. Rus. Bakebot: Baking cookies with the pr2. In *IEEE/RSJ International Conference on Intelligent Robots and Systems PR2 Workshop*, 2011.

[10] C. Crick, S. Osentoski, G. Jay, and O. C. Jenkins.

Fig. 14. **Examples of objects and object parts.** Each image shows the point cloud representation of an object. We overlaid some of its parts by CAD models for online Robobarista crowd-sourcing platform. Note that the actual underlying point-cloud of object parts contains much more noise and is not clearly segmented, and none of the models have access to overlaid model for inferring manipulation trajectory.

Human and robot perception in large-scale learning from demonstration. In *HRI*. ACM, 2011.

[11] H. Dang and P. K. Allen. Semantic grasping: Planning robotic grasps functionally suitable for an object manipulation task. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[12] C. Daniel, G. Neumann, and J. Peters. Learning concurrent motor skills in versatile solution spaces. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012.

[13] R. Detry, C. H. Ek, M. Madry, and D. Kragic. Learning a dictionary of prototypical grasp-predicting parts from grasping experience. In *IEEE International Conference on Robotics and Automation*, 2013.

[14] F. Endres, J. Trinkle, and W. Burgard. Learning the dynamics of doors for robotic manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[15] G. Erdogan, I. Yildirim, and R. A. Jacobs. Transfer of object shape knowledge across visual and haptic modalities. In *Proceedings of the 36th Annual Conference of the Cognitive Science Society*, 2014.

[16] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *PAMI*, 32(9):1627–1645, 2010.

[17] M. Forbes, M. J.-Y. Chung, M. Cakmak, and R. P. Rao. Robot programming by demonstration with crowdsourced action fixes. In *Second AAAI Conference on Human Computation and Crowdsourcing*, 2014.

[18] J. J. Gibson. *The ecological approach to visual perception*. Psychology Press, 1986.

[19] R. Girshick, P. Felzenszwalb, and D. McAllester. Object detection with grammar models. In *NIPS*, 2011.

[20] I. Goodfellow, Q. Le, A. Saxe, H. Lee, and A. Y. Ng. Measuring invariances in deep networks. In *NIPS*, 2009.

[21] R. Hadsell, A. Erkan, P. Sermanet, M. Scoffier, U. Muller, and Y. LeCun. Deep belief net learning in a long-range vision system for autonomous off-road driving. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 628–633. IEEE, 2008.

[22] G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313 (5786):504–507, 2006.

[23] K. Hsiao, S. Chitta, M. Ciocarlie, and E. Jones. Contact-reactive grasping of objects with partial shape information. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[24] J. Hu, J. Lu, and Y.-P. Tan. Discriminative deep metric learning for face verification in the wild. In *The IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[25] N. Hu, Z. Lou, G. Englebienne, and B. Krse. Learning to recognize human activities from soft labeled data. In *Proceedings of Robotics: Science and Systems*, Berkeley, USA, July 2014.

[26] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, et al. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *ACM Symposium on UIST*, 2011.

[27] A. Jain, B. Wojcik, T. Joachims, and A. Saxena. Learning preferences for manipulation tasks from online coactive feedback. In *International Journal of Robotics Research*, 2015.

[28] T. Joachims, T. Finley, and C.-N. J. Yu. Cutting-plane training of structural svms. *Machine Learning*, 2009.

[29] D. Katz, M. Kazemi, J. Andrew Bagnell, and A. Stentz. Interactive segmentation, tracking, and kinematic modeling of unknown 3d articulated objects. In *IEEE International Conference on Robotics and Automation*, pages 5003–5010. IEEE, 2013.

[30] B. Kehoe, A. Matsukawa, S. Candido, J. Kuffner, and K. Goldberg. Cloud-based robot grasping with the google object recognition engine. In *IEEE International Conference on Robotics and Automation*, 2013.

[31] H. Koppula and A. Saxena. Anticipating human activities using object affordances for reactive robotic response. In *Robotics: Science and Systems*, 2013.

[32] H. Koppula, A. Anand, T. Joachims, and A. Saxena. Semantic labeling of 3d point clouds for indoor scenes. *NIPS*, 2011.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[34] O. Kroemer, E. Ugur, E. Oztop, and J. Peters. A kernel-based approach to direct action perception. In *IEEE International Conference on Robotics and Automation*, 2012.

[35] K. Lai, L. Bo, and D. Fox. Unsupervised feature learning for 3d scene labeling. In *IEEE International Conference on Robotics and Automation*, 2014.

[36] I. Lenz, H. Lee, and A. Saxena. Deep learning for detecting robotic grasps. *Robotics: Science and Systems*, 2013.

[37] I. Lenz, R. Knepper, and A. Saxena. Deepmpc: Learning deep latent features for model predictive control. In *Robotics: Science and Systems*, 2015.

[38] S. Levine, N. Wagener, and P. Abbeel. Learning contact-rich manipulation skills with guided policy search. *IEEE International Conference on Robotics and Automation*, 2015.

[39] L.-J. Li, R. Socher, and L. Fei-Fei. Towards total scene understanding: Classification, annotation and segmentation in an automatic framework. In *The IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[40] O. Mangin, P.-Y. Oudeyer, et al. Unsupervised learning of simultaneous motor primitives through imitation. In *IEEE ICDL-EPIROB*, 2011.

[41] T. Mikolov, Q. V. Le, and I. Sutskever. Exploiting similarities among languages for machine translation. *CoRR*, 2013.

[42] S. Miller, J. Van Den Berg, M. Fritz, T. Darrell, K. Goldberg, and P. Abbeel. A geometric approach to robotic laundry folding. *International Journal of Robotics Research*, 2012.

[43] D. Misra, J. Sung, K. Lee, and A. Saxena. Tell me dave: Context-sensitive grounding of natural language to mobile manipulation instructions. In *Robotics: Science and Systems*, 2014.

[44] J. Moore, S. Chen, T. Joachims, and D. Turnbull. Learning to embed songs and tags for playlist prediction. In *Conference of the International Society for Music Information Retrieval (ISMIR)*, pages 349–354, 2012.

[45] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *PAMI*, 2014.

[46] K. Mülling, J. Kober, O. Kroemer, and J. Peters. Learning to select and generalize striking movements in robot table tennis. *International Journal of Robotics Research*, 32(3):263–279, 2013.

[47] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng. Multimodal deep learning. In *International Conference on Machine Learning*, 2011.

[48] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal. Learning and generalization of motor skills by learning from demonstration. In *IEEE International Conference on Robotics and Automation*, 2009.

[49] M. Phillips, V. Hwang, S. Chitta, and M. Likhachev. Learning to plan for constrained manipulation from demonstrations. In *Robotics: Science and Systems*, 2013.

[50] S. Pillai, M. Walter, and S. Teller. Learning articulated motions from visual demonstration. In *Robotics: Science and Systems*, 2014.

[51] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5, 2009.

[52] R. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation*, 2011.

[53] K. Shoemake. Animating rotation with quaternion curves. *SIGGRAPH*, 19(3):245–254, 1985.

[54] R. Socher, J. Pennington, E. Huang, A. Ng, and C. Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *EMNLP*, 2011.

[55] R. Socher, B. Huval, B. Bhat, C. Manning, and A. Ng. Convolutional-recursive deep learning for 3d object classification. In *NIPS*, 2012.

[56] K. Sohn, W. Shang, and H. Lee. Improved multimodal deep learning with variation of information. In *NIPS*, 2014.

[57] N. Srivastava and R. R. Salakhutdinov. Multimodal learning with deep boltzmann machines. In *NIPS*, 2012.

[58] J. Sturm, C. Stachniss, and W. Burgard. A probabilistic framework for learning kinematic models of articulated objects. *Journal of Artificial Intelligence Research*, 41 (2):477–526, 2011.

[59] J. Sung, C. Ponce, B. Selman, and A. Saxena. Un-structured human activity detection from rgbd images. In *IEEE International Conference on Robotics and Automation*, 2012.

[60] J. Sung, B. Selman, and A. Saxena. Synthesizing manipulation sequences for under-specified tasks using unrolled markov random fields. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[61] J. Sung, S. H. Jin, and A. Saxena. Robobarista: Object part-based transfer of manipulation trajectories from crowd-sourcing in 3d pointclouds. In *International Symposium on Robotics Research (ISRR)*, 2015.

[62] S. Tellex, R. Knepper, A. Li, T. Howard, D. Rus, and N. Roy. Asking for help using inverse semantics. *Robotics: Science and Systems*, 2014.

[63] J. Tenenbaum, V. De Silva, and J. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[64] S. Thrun, W. Burgard, D. Fox, et al. *Probabilistic robotics*. MIT press Cambridge, 2005.

[65] R. Toris and S. Chernova. Robotsfor. me and robots for you. In *Proceedings of the Interactive Machine Learning Workshop, Intelligent User Interfaces Conference*, pages 10–12, 2013.

[66] R. Toris, D. Kent, and S. Chernova. The robot management system: A framework for conducting human-robot interaction studies through crowdsourcing. *Journal of Human-Robot Interaction*, 3(2):25–49, 2014.

[67] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *ICML*. ACM, 2004.

[68] I. Tsochantaridis, T. Joachims, T. Hofmann, Y. Altun, and Y. Singer. Large margin methods for structured and interdependent output variables. *JMLR*, 6(9), 2005.

[69] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9 (2579-2605):85, 2008.

[70] F. Vina, Y. Bekiroglu, C. Smith, Y. Karayiannidis, and D. Kragic. Predicting slippage and learning manipulation affordances through gaussian process regression. In *Humanoids*, 2013.

[71] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning*, 2008.

[72] K. Q. Weinberger, J. Blitzer, and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. In *NIPS*, 2005.

[73] J. Weston, S. Bengio, and N. Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *IJCAI*, 2011.

[74] S. Wieland, D. Gonzalez-Aguirre, N. Vahrenkamp, T. Asfour, and R. Dillmann. Combining force and visual feedback for physical interaction tasks in humanoid robots. In *Humanoid Robots*, 2009.

[75] C. Wu, I. Lenz, and A. Saxena. Hierarchical semantic labeling for task-relevant rgb-d perception. In *Robotics:*

Science and Systems, 2014.

[76] C.-N. Yu and T. Joachims. Learning structural svms with latent variables. In *International Conference on Machine Learning*, 2009.

[77] M. D. Zeiler. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[78] M. D. Zeiler, M. Ranzato, R. Monga, et al. On rectified linear units for speech processing. In *ICASSP*, 2013.