

Deep Multimodal Embedding: Manipulating Novel Objects with Point-clouds, Language and Trajectories

Jaeyong Sung^{1,2}, Ian Lenz¹ and Ashutosh Saxena³

Abstract—A robot operating in a real-world environment needs to perform reasoning over a variety of sensor modalities such as vision, language and motion trajectories. However, it is extremely challenging to manually design features relating such disparate modalities. In this work, we introduce an algorithm that learns to embed point-cloud, natural language, and manipulation trajectory data into a shared embedding space with a deep neural network. To learn semantically meaningful spaces throughout our network, we use a loss-based margin to bring embeddings of relevant pairs closer together while driving less-relevant cases from different modalities further apart. We use this both to pre-train its lower layers and fine-tune our final embedding space, leading to a more robust representation. We test our algorithm on the task of manipulating novel objects and appliances based on prior experience with other objects. On a large dataset, we achieve significant improvements in both accuracy and inference time over the previous state of the art. We also perform end-to-end experiments on a PR2 robot utilizing our learned embedding space.

I. INTRODUCTION

Consider a robot manipulating a new appliance in a home kitchen, *e.g.* the toaster in Figure 3. The robot must use the combination of its observations of the world and natural language instructions to infer how to manipulate objects. Such ability to fuse information from different input modalities and map them to actions is extremely useful to many applications of household robots [1], including assembling furniture, cooking recipes, and many more.

Even though similar concepts might appear very differently in different sensor modalities, humans are able to understand that they map to the same concept. For example, when asked to “turn the knob counter-clockwise” on a toaster, we are able to correlate the instruction language and the appearance of a knob on a toaster with the motion to do so. We also associate this concept more closely with a motion which would incorrectly rotate in the opposite direction than with, for example, the motion to press the toaster’s handle downwards. There is strong evidence that humans are able to correlate between different modalities through *common representations* [2].

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, here our trajectory representation is inherently dense, while our representation of language is naturally sparse. This makes it challenging to apply algorithms designed for unimodal data. Second, even

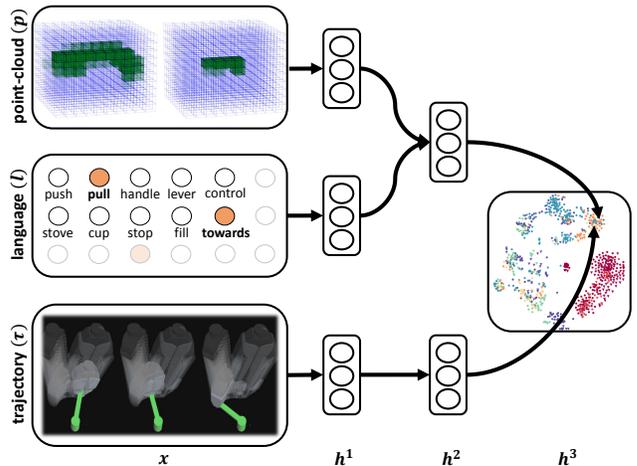


Fig. 1. **Deep Multimodal Embedding:** Our deep neural network learns to embed both point-cloud/natural language instruction combinations and manipulation trajectories in the same semantically meaningful space, where distance represents the relevance of embedded data.

with expert knowledge, it is extremely challenging to design joint features between such disparate modalities. Designing features which map different sensor inputs and actions to the same space, as required here, is particularly challenging.

In this work, we use a deep neural network to learn a shared embedding between the pairing of object parts in the environment with natural language instructions, and manipulation trajectories (Figure 1). 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 they are. Our algorithm allows for efficient inference because, given a new instruction and point-cloud, we need only find the nearest trajectory to the projection of this pair in the learned embedding space, which can be done using fast nearest-neighbor algorithms [3].

In the past, deep learning methods have shown impressive results for learning features for a wide variety of domains [4], [5], [6] and even learning cross-domain embeddings [7]. 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. 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 [8] or train for reconstruction [9].

¹Department of Computer Science, Cornell University. ²Department of Computer Science, Stanford University. ³Brain Of Things, Inc. {jysung, ianlenz, asaxena}@cs.cornell.edu

In order to validate our approach, we test our model on a large manipulation dataset, the Robobarista Dataset [1], which focuses on learning to infer manipulation trajectories for *novel* objects. We also present results on a series of experiments on a PR2 robot, showing that our algorithm is able to successfully manipulate several different objects. In summary, the key contributions of this work are:

- We present an algorithm which learns a semantically meaningful embedding space by enforcing a varying and loss-based margin.
- We present an algorithm for unsupervised pre-training of multi-modal features to be used for embedding which outperforms standard pre-training algorithms [9].
- We present a novel approach to manipulation via embedding multimodal data with deep neural networks.
- Our approach allows for fast inference of manipulation trajectories for novel objects, roughly 170x faster than previous work [1] while also improving accuracy on a large manipulation dataset [1].

II. RELATED WORK

A. Metric Embedding

Several works in machine learning make use of the power of shared embedding spaces. Large margin nearest neighbors (LMNN) [10] learns a max-margin Mahalanobis distance for a unimodal input feature space. [11] learn linear mappings from image and language features to a common embedding space for automatic image annotation. [12] 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.

B. Deep Learning

Multimodal data: A deep neural network has been used to learn features of video and audio [13]. With a generative learning, a network can be robust to missing modalities at inference time [14]. In these works, similar to [1], a single network takes all modalities as inputs, whereas here we perform joint embedding of multiple modalities using multiple networks.

Joint embedding: Several works use deep networks for joint embedding between different feature spaces. For translation, a joint feature space is learned from different languages [15]; for annotation and retrieval, images and natural language “tags” are mapped to the same space [7]. We present a new pre-training algorithm for embedding spaces and show that it outperforms the conventional methods used in these works.

A deep network is also used as metric learning for the face verification task [16], which enforces a constant margin between distances among inter-class objects and among intra-class objects, similar to LMNN [10]. In Sec. VI-A, we show that our approach, which uses a loss-dependent variable margin, produces better results for our problem.

C. Robotic Manipulation

Many works in robotic manipulation focus on task-specific manipulation with *known* objects — for example, folding towels [17], baking cookies [18], or planar contact manipulation [19]. Others [20], [21] focus on sequencing manipulation tasks or choosing when to switch skills [22], assuming manipulation primitives such as *pour* are available. For novel objects, affordances are predicted and associated motions are applied [23]. Instead, similar to [1], we skip intermediate representations and directly generalize to novel objects.

A few recent works use deep learning approaches for robotic manipulation. Deep networks have been used to detect stable grasps from RGB-D data [24], [25]. [26] use a Gaussian mixture model to learn system dynamics, then use these to learn a manipulation policy using a deep network. [27] use a deep network to learn system dynamics for real-time model-predictive control. Both these works focus on learning low-level input-output controllers. Here, we instead focus on inferring full 6-DoF trajectories, which such controllers could then be used to follow.

Sung et al. [1] perform object part-based transfer of manipulation trajectories for novel objects and introduces a large manipulation dataset including objects like espresso machine and urinal. We primarily test our algorithm on this dataset. In Sec. VI-A, we show that our approach gives better accuracy than this prior work, while also running 171x faster.

III. OVERVIEW

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. In particular, we focus on point-clouds of object parts, natural language instructing manipulation of different objects, and trajectories that would manipulate these objects.

We introduce a method that learns a common point-cloud/language/trajectory embedding space in which the projection of a task (point-cloud/language combination) should have a higher similarity to projections of relevant trajectories 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, with 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.

We learn non-linear embeddings using a deep learning approach, as shown in Fig. 1, which maps raw data from these three different modalities to a joint embedding space. 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.

We show in Sec. V that a learned joint embedding space can be efficiently used for finding an appropriate manipulation trajectory for objects with natural language instructions.

A. Problem Formulation

Given tuples of a scene $p \in \mathcal{P}$, a natural language instruction $l \in \mathcal{L}$ and an end-effector trajectories $t \in \mathcal{T}$, our goal is to learn a joint embedding space and two different mapping functions that map to this 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 :

$$\begin{aligned} \Phi_{\mathcal{P},\mathcal{L}}(p, l) : (\mathcal{P}, \mathcal{L}) &\rightarrow \mathbb{R}^M \\ \Phi_{\mathcal{T}}(\tau) : \mathcal{T} &\rightarrow \mathbb{R}^M \end{aligned}$$

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

IV. LEARNING JOINT POINT-CLOUD/LANGUAGE/TRAJECTORY MODEL

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 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 that 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 fully manipulate the object according to the instruction.

For every training point-cloud/language pair (p_i, l_i) , we have a set of demonstrations \mathcal{T}_i and the most optimal demonstration trajectory $\tau_i^* \in \mathcal{T}_i$. Using the optimal demonstration τ_i^* 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 use the DTW-MT distance function (described later in Sec. VI) as 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 a strategy previously used for handling noise in crowd-sourced data [1], we can use thresholds (t_S, t_D) to generate two sets from the pool of all trajectories:

$$\begin{aligned} \mathcal{T}_{i,S} &= \{\tau \in \mathcal{T} \mid \Delta(\tau_i^*, \tau) < t_S\} \\ \mathcal{T}_{i,D} &= \{\tau \in \mathcal{T} \mid \Delta(\tau_i^*, \tau) > t_D\} \end{aligned}$$

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 [10], which can be written in the form of a constraint:

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

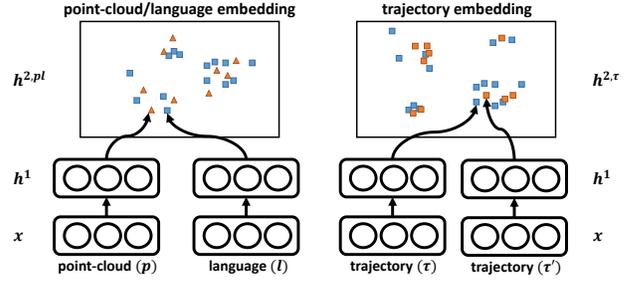


Fig. 2. **Pre-training lower layers:** Visualization of our pre-training approaches for $h^{2,pl}$ and $h^{2,\tau}$. For $h^{2,pl}$, our algorithm pushes matching point-clouds and instructions to be more similar. For $h^{2,\tau}$, our algorithm pushes trajectories with higher DTW-MT similarity to be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

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 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:

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

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 [28], 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 value after the similarity is augmented with the loss scaled by a constant α :

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

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

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

The average cost of each minibatch is back-propagated through all the layers of the deep neural network using the AdaDelta [29] 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

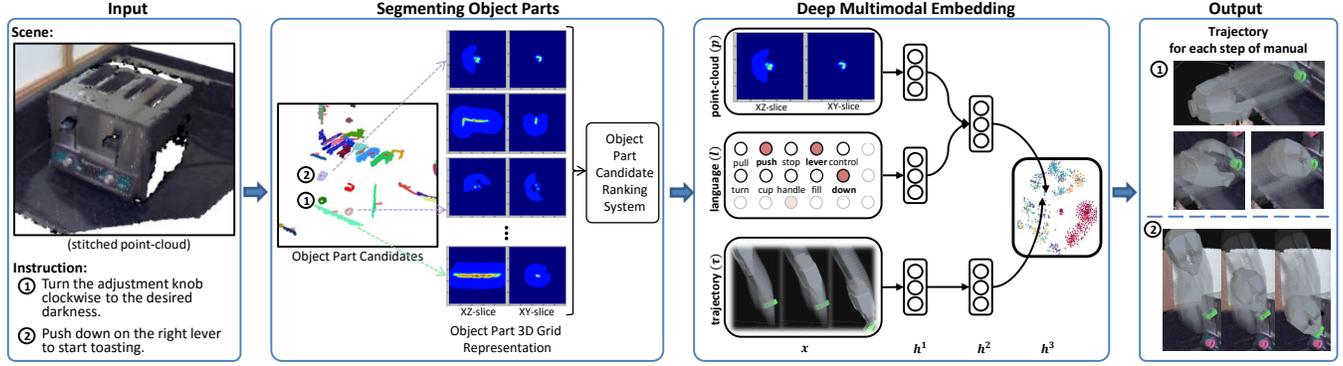


Fig. 3. **System Overview:** Given a point-cloud and a language instruction, our goal is to output a trajectory that would manipulate the object according to the instruction. The given point-cloud scene is segmented into many parts and ranked for each step of the instruction manual. By embedding point-cloud, language, and trajectory modalities into a joint embedding space, our algorithm selects the best trajectory to transfer to the new object.

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 VI-A.

First, we pre-train the layers leading up to these layers using sparse de-noising autoencoders [30], [31]. 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:

$$l' = \operatorname{argmax}_{l \in \mathcal{L}} (\operatorname{sim}(\Phi_{\mathcal{P}}(p_i), \Phi_{\mathcal{L}}(l)) + \alpha \Delta(\tau_i, \tau))$$

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

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 denoising autoencoder [30], [31] would only pre-train $h^{2,\tau}$ to reconstruct individual trajectories. Instead, we employ pre-training similar to pre-training of $h^{2,pl}$ above, except now we pre-train for only a single modality — trajectory data.

As shown on right hand side of Fig. 2, the layer that embeds to $h^{2,\tau}$ is duplicated. These 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 minimize a similar cost function as we do for $h^{2,pl}$.

V. APPLICATION: MANIPULATING NOVEL OBJECTS

As an example application, we consider manipulating novel appliances [1]. Our goal is to use our learned embedding space to allow the robot to infer a manipulation trajectory when it is introduced to a new appliance with its natural language instruction manual. For example, as shown in Fig. 3, given a point-cloud of a scene with a toaster and an instruction such as ‘Push down on the right lever to start toasting,’ it should output a trajectory, representative of how a two-fingered end-effector should move, including how to approach, grasp, and push down on the lever. Our algorithm allows the robot to leverage prior experience with *different* appliances — for example, a trajectory which manipulates the handles of a paper towel dispenser might be transferred to manipulate the toaster handle.

First, in order to correctly identify a part p out of a scene s that an instruction asks to manipulate, a point-cloud of a scene s is segmented into many small potential candidates. All segments are ranked for each step of the manual instruction. Multiple variations of correct segmentations and lots of incorrect segmentation make our embedding representation even more robust as shown later in Sec. VI-A.

Then, from a library of trajectories with prior experience, the trajectory that gives the highest similarity to the selected point-cloud p and language l in our embedding space \mathbb{R}^M :

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

As in [11], similarity is defined as $\operatorname{sim}(a, b) = a \cdot b$.

The previous approach to this problem [1] requires projecting a new point-cloud and natural language instruction with *every* trajectory in the training set exhaustively through the network during inference.

Instead, our approach allows us to pre-embed all candidate trajectories into a shared embedding space. The correct trajectory can be identified by embedding only a new point-cloud/language pair. As shown in Sec. VI-A, this significantly improves both the inference run-time and accuracy and makes it much more scalable to a larger training dataset.

A. Segmenting Object Parts from Point-clouds

Our learning algorithm (Sec. IV) assumes that object parts p corresponding to each instruction l have already been segmented from the point-cloud scene s . While our focus is on learning to manipulate these segmented parts, we also introduce a segmentation approach which allows us to both build an end-to-end system and augment our training data for better unsupervised learning, as shown in Sec. VI-A.

1) *Generating Object Part Candidates*: We employ a series of geometric feature based techniques to segment a scene s into small overlapping segments $\{p_1, p_2, \dots, p_n\}$. We first extract Euclidean clusters of points while limiting the difference of normals between a local and larger region [32], [33]. We then filter out segments which are too big for human hands. To handle a wide variety of object parts of different scales, we generate two sets of candidates with two different sets of parameters, which are combined for evaluation.

2) *Part Candidate Ranking Algorithm*: Given a set of segmented parts p , we must now use our training data \mathcal{D} to select for each instruction l_j the best-matching part p_j^* . We do so by optimizing the score $\psi(p_j, l_j)$ of each segment p_i for a step l_j in a manual, evaluated in three parts:

$$\psi(p_i, l_j; \mathcal{D}) = \psi_{feat}(p_i, l_j)(\psi_{pc}(p_i, l_j) + \psi_{lang}(p_i, l_j))$$

The score ψ_{pc} is based on the k_p -most identical segments from the training data \mathcal{D} , based on cosine similarity using our grid representation (Sec. V-B). The score is a sum of similarity against these segments and their associated language: $\psi_{pc}(p_i, l_j) = \sum_{k=1}^n (\text{sim}(p_i, p_k) + \beta \text{sim}(l_j, l_k))$. If the associated language does not exist (*i.e.* p_k is not a manipulable part), it is given a set similarity value. Similarly, the score ψ_{lang} is based on the k_l -most identical language instructions in \mathcal{D} . It is a sum of similarity against identical language and associated expert point-cloud segmentations.

The feature score ψ_{feat} is computed by weighted features $w^T \phi_{feat}(p_i, l_j)$ as described in Sec. V-A.3. Each score of the segmented parts $\psi(p_i, l_j)$ is then adjusted by multiplying by ratio of its score against the marginalized score in the manual: $\hat{\psi}(p_i, l_j) = \frac{\psi(p_i, l_j)}{\sum_{l_k \in m_{new}} \psi(p_i, l_k)} \psi(p_i, l_j)$. For each $l_j \in m_{new}$, an optimal segment of the scene chosen as the segment with the maximum score: $\max_{p_i \in s_{new}} \hat{\psi}(p_i, l_j)$.

3) *Features*: Three features are computed for each segment in the context of the original scene. We first infer where a person would stand by detecting the ‘front’ of the object, by a plane segmentation constrained to have a normal axis less than 45° from the line between the object’s centroid and the original sensor location, assuming that the robot is introduced close to the ‘front’. We then compute a ‘reach’ distance from an imaginary person 170cm tall, which is defined as the distance from the head of the person to each segment subtracted by the distance of the closest one. Also, because stitched point-clouds have significant noise near their edges, we compute the distance from the imaginary view ray, a line defined by the centroid of the scene to the head of the person. Lastly, objects like a soda fountain a sauce dispenser have many identical parts, making it difficult to disambiguate different choices (e.g. Coke/Sprite, ketchup/mustard). Thus,

for such scenarios, we also provided a 3D point as if human is pointing at the label of the desired selection. Note that this does not point at the actual part but rather at its label or vicinity. A distance from this point is also used as a feature.

B. Data Representation

All three data modalities (p, l, τ) are variable-length and must be transformed into a fixed-length representation.

Each point-cloud segment is converted into a real-valued 3D occupancy grid where each cell’s value is proportional to how many points fall into the cube it spans. We use a $100 \times 100 \times 100$ grid of cubic cells with sides of 0.25cm . Unlike our previous work [1], each cell count is also distributed to the neighboring cells with an exponential distribution. This smooths out missing points and increases the amount of information represented. The grid then is normalized to be between $0 \sim 1$ by dividing by the maximal count.

While our approach focuses on the shape of the part in question, the shape of the nearby scene can also have a significant effect on how the part is manipulated. To account for this, we assign a value of 0.2 to any cell which only contains points which belong to the scene but not the specific part in question, but are within some distance from the nearest point for the given part. To fill hollow parts behind the background, such as tables and walls, we ray-trace between the starting location of the sensor and cells filled by background points and fill these similarly.

While our segment ranking algorithm uses the full-sized grid for each segment, our main embedding algorithm uses two compact grids generated by taking average of cells: $10 \times 10 \times 10$ grids with cells with sides of 2.5cm and of 1cm .

Natural language l is represented by a fixed-size bag-of-words. Trajectories are variable-length sequences of waypoints, where each waypoint contains a position, an orientation and a gripper state (‘open’/‘closed’/‘holding’) defined in the coordinate frame of object part. All trajectories τ are then normalized to a fixed length of 15 waypoints. For more details on trajectory representation, please refer to [1].

C. Robotic Platform

We tested our algorithms on a PR2, a mobile robot with two arms with seven degrees of freedom each (Fig. 5). All software for controlling the robot is written in ROS [34], and the embedding algorithm are written with Theano [35]. All of the computations are done on a remote computer utilizing a GPU for our embedding model.

D. Parameters

Through validation, we found an optimal embedding space size M of 25 and intermediate-layer sizes $N_{1,p}$, $N_{1,l}$, $N_{1,\tau}$, $N_{2,pl}$, and $N_{2,\tau}$ of 250, 150, 100, 125, and 100 with the loss scaled by $\alpha = 0.2$. These relatively small layer sizes also had the advantage of fast inference, as shown in Sec. VI-A.

VI. EXPERIMENTS

In this section, we present two sets of experiments. First, we show offline experiments on the Robobarista dataset [1] which test individual components of our system, showing

TABLE I

RESULT ON ROBOBARISTA DATASET WITH 5-fold cross-validation.

ROWS LIST MODELS WE TESTED INCLUDING OUR MODEL AND BASELINES. EACH COLUMN SHOWS A DIFFERENT METRIC USED TO EVALUATE THE MODELS. FOR THE DTW-MT METRIC, LOWER VALUES ARE BETTER. FOR ACCURACY, HIGHER IS BETTER.

Models	per manual		per instruction	
	DTW-MT	DTW-MT	DTW-MT	Accu. (%)
<i>chance</i>	28.0 (± 0.8)	27.8 (± 0.6)	11.2 (± 1.0)	
<i>object part classifier</i> [1]	-	22.9 (± 2.2)	23.3 (± 5.1)	
<i>LSSVM + kinematic</i> [1]	17.4 (± 0.9)	17.5 (± 1.6)	40.8 (± 2.5)	
<i>similarity + weights</i> [1]	13.3 (± 1.2)	12.5 (± 1.2)	53.7 (± 5.8)	
<i>Sung et al.</i> [1]	13.0 (± 1.3)	12.2 (± 1.1)	60.0 (± 5.1)	
<i>LMNN-like cost</i> [10]	15.4 (± 1.8)	14.7 (± 1.6)	55.5 (± 5.3)	
<i>Ours w/o pretraining</i>	13.2 (± 1.4)	12.4 (± 1.0)	54.2 (± 6.0)	
<i>Ours with SDA</i>	11.5 (± 0.6)	11.1 (± 0.6)	62.6 (± 5.8)	
<i>Ours w/o Mult. Seg</i>	11.0 (± 0.8)	10.5 (± 0.7)	65.1 (± 4.9)	
<i>Our Model</i>	10.3 (± 0.8)	9.9 (± 0.5)	68.4 (± 5.2)	

improvements for each. Second, we present a series of real-world robotic experiments which show that our system is able to produce trajectories that can successfully manipulate objects based on natural language instructions.

Dataset. We test our model on the Robobarista dataset [1]. This dataset consists of 115 point-cloud scenes with 154 manuals, consisting of 249 expert segmented point-clouds and 250 free-form natural language instructions. It also contains 1225 crowd-sourced manipulation trajectories which are demonstrated for 250 point-cloud/language pairs. The point-clouds are collected by stitching multiple views using Kinect Fusion. The manipulation trajectories are collected from 71 non-experts on Amazon Mechanical Turk.

Evaluation. All algorithms are evaluated using *five-fold cross-validation*, with 10% of the data kept out as a validation set. For each point-cloud/language pair in test set of each fold, each algorithm chooses one trajectory from the training set which best suits this pair. Since our focus is on testing ability to reason about different modalities and transfer trajectory, segmented parts are provided as input. To evaluate transferred trajectories, the dataset contains a separate expert demonstration for each point-cloud/language pair, which is not used in the training phase [1]. Every transferred trajectory is evaluated against these expert demonstrations.

Metrics. For evaluation of trajectories, we use dynamic time warping for manipulation trajectories (DTW-MT) [1], which non-linearly warps two trajectories of different lengths while preserving weak ordering of matched trajectory waypoints. Since its values are not intuitive, [1] also reports the percentage of transferred trajectories that have a DTW-MT value of less than 10 from the ground-truth trajectory, which indicates that it will most likely correctly manipulate according to the given instruction according to an expert survey.

We report three metrics: DTW-MT per manual, DTW-MT per instruction, and Accuracy (DTW-MT < 10) per instruction. Instruction here refers to every point-cloud/language pair, and manual refers to list of instructions which comprises a set of sequential tasks, which we average over.

Baselines. We compare our model against several baselines:

- 1) *Chance*: Trajectories are randomly transferred.
- 2) *Sung et al. [1]*: State-of-the-art result on this dataset

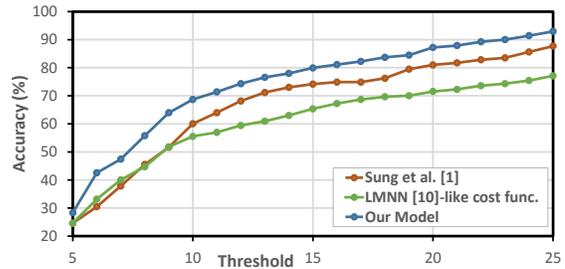


Fig. 4. Accuracy-threshold graph showing results of varying thresholds on DTW-MT measures. Our algorithm consistently outperforms the previous approach [1] and an LMNN-like cost function [10].

that trains a neural network to predict how likely each known trajectory matches a given point-cloud and language.

We also report several baselines from this work which rely on more traditional approaches such as classifying point-clouds into labels like ‘handle’ and ‘lever’ (*object part classifier*), or hand-designing features for multi-modal data (*LSSVM + kinematic structure, task similarity + weighting*).

3) *LMNN [10]-like cost function*: For both fine-tuning and pre-training, we define the cost function without loss augmentation. Similar to LMNN [10], we give a finite margin between similarities. For example, as cost function for h^3 :

$$|1 + \text{sim}(\Phi_{\mathcal{P}, \mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau')) - \text{sim}(\Phi_{\mathcal{P}, \mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau_i))|_+$$

4) *Our Model without Pretraining*: Our full model fine-tuned without any pre-training of lower layers.

5) *Our Model with SDA*: Instead of pre-training $h^{2,pl}$ and $h^{2,\tau}$ as defined in Secs. IV-A and IV-B, we pre-train each layer as stacked de-noising autoencoders [30], [31].

6) *Our Model without Multiple Segmentations*: Our model trained only with expert segmentations, without taking utilizing all candidate segmentations in auto-encoders and multiple correct segmentations of the same part during training.

A. Results

We present the results of our algorithm and the baseline approaches in Table I. Additionally, Fig. 4 shows accuracies obtained by varying the threshold on the DTW-MT measure.

The state-of-the-art result [1] on this dataset has a DTW-MT measure of 13.0 per manual and a DTW-MT measure and accuracy of 12.2 and 60.0% per instruction. Our full model based on joint embedding of multimodal data achieved 10.3, 9.9, and 68.4%, respectively. This means that when a robot encounters a *new* object it has never seen before, our model gives a trajectory which would correctly manipulate it according to a given instruction approximately 68.4% of the time. From Fig. 4, we can see that our algorithm consistently outperforms both prior work and an LMNN-like cost function for all thresholds on the DTW-MT metric.

What does our learned deep embedding space represent? Fig. 6 shows a visualization of the top layer h^3 , the joint embedding space. 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 [36].

One interesting result is that our system was able to naturally learn that “nozzle” and “spout” are effectively



Fig. 5. **Robotic Experiments:** We test our algorithm on a PR2 robot with three different novel objects — coffee dispenser handle, beverage dispenser lever, and door handle.

synonyms for purposes of manipulation. It clustered these together in the upper-left of Fig. 6 based solely on the fact that both are associated with similar point-cloud shapes and manipulation trajectories.

In addition to the aforementioned cluster, 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 2 o’clock and 8 o’clock in Fig. 6, respectively. Despite the fact that 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 the cost function be loss-augmented? When we change 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 [10], performance drops to 55.5%. This loss-augmentation is also visible in our embedding space. Notice the purple cluster around the 12 o’clock region of Fig. 6, and the right portion of the red cluster in the 11 o’clock region. The purple cluster represents tasks and demonstrations related to pushing a bar, 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 [30], [31] increases performance to 62.6%, still significantly underperforming our pre-training algorithm, at 68.4%. This shows that our metric embedding pre-training approach provides a better initialization for an embedding space than SDA.

Can automatically segmented object parts be manipulated? From Table II, we see that our segmentation approach was able to find a good segmentation for the object parts in question in 50 of 60 robotic trials (Sec. VI-B), or 83.3% of the time. Most failures occurred for the beverage dispenser, which had a small lever that was difficult to segment.

When our full DME model utilizes two variations of same part and uses all candidates as a training data for the auto-encoder, our model performs at 68.4% compared to 65.1% which only used expert segmentations.

Does embedding improve efficiency? While [1] has 749,638 parameters to be learned, our model only has 616,175 (and still gives better performance.)

The previous model had to compute joint point-

TABLE II

RESULTS OF 60 EXPERIMENTS ON A PR2 ROBOT RUNNING END-TO-END EXPERIMENTS AUTONOMOUSLY ON THREE DIFFERENT OBJECTS.

Success Rate of Each Step	Dispenser Handle	Beverage Lever	Door Handle	Avg.
1) Segmentation	90.0%	65.0%	95.0%	83.3%
2) DME Traj. Inference	94.4%	100.0%	78.9%	91.1%
3) Execution of Traj.	82.4%	76.9%	100.0%	86.4%

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 Theano [35]. We measured inference times 10000 times for first test fold, which has a pool of 962 trajectories. Times to preprocess the data and load into GPU memory were not included in this measurement.

B. Robotic Experiments

To test our framework, we performed 60 experiments on a PR2 robot in three different scenes shown in Fig. 5. We presented the robot with the object placed within reach from different starting locations along with a language instruction.

Table II shows results of our robotic experiments. It was able to successfully complete the task end-to-end autonomously 39 times. Segmentation was not as reliable for the beverage dispenser which has a small lever. However, when segmentation was successful, our embedding algorithm was able to provide a correct trajectory with an accuracy of 91.1%. Our PR2 was then able to correctly follow these trajectories with a few occasional slips due to the relatively large size of its hand compared to the objects.

Video of robotic experiments are available at this website: <http://www.robobarista.org/dme/>

VII. CONCLUSION

In this work, we introduce an algorithm that learns a semantically meaningful common embedding space for three modalities — point-cloud, natural language and trajectory. Using a loss-augmented cost function, we learn to embed in a joint space such that similarity of any two points in the space reflects how relevant to each other. As one of application, we test our algorithm on the problem of manipulating novel objects. We empirically show on a large dataset that our embedding-based approach significantly improve accuracy, despite having less number of learned parameters and being much more computationally efficient (about 171x faster) than the state-of-the-art result. We also show via series of robotic experiments that our segmentation algorithm and embedding algorithm allows robot to autonomously perform the task.

Acknowledgment. This work was supported by Microsoft Faculty Fellowship and NSF Career Award to Saxena.

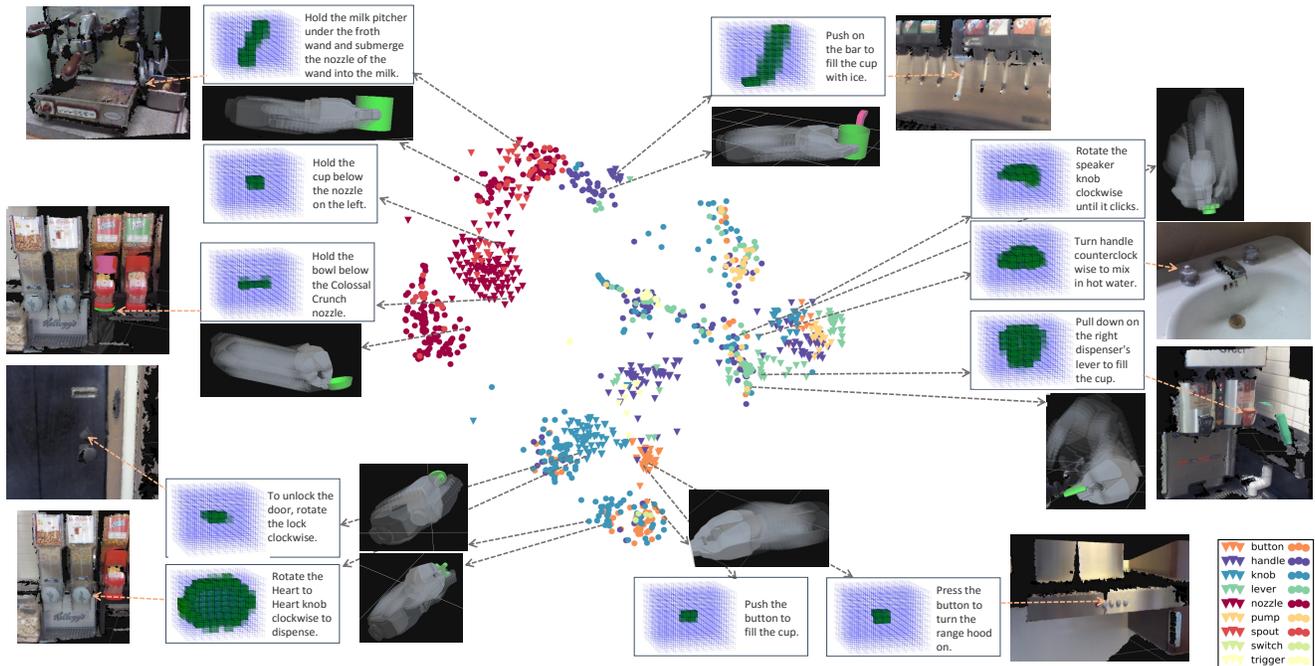


Fig. 6. **Learned Deep Point-cloud/Language/Trajectory Embedding Space:** Joint embedding space h^3 after the network is fully fine-tuned, visualized in 2d using t-SNE [36]. Inverted triangles represent projected point-cloud/language pairs, circles represent projected trajectories. The occupancy grid representation (Sec. V-B) of object part point-clouds is shown in green in blue grids. For presentation purpose, ‘neighbor’ cells are not shown. 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. These figures are best viewed in color.

REFERENCES

- [1] J. Sung, S. H. Jin, and A. Saxena, “Robobarista: Object part-based transfer of manipulation trajectories from crowd-sourcing in 3d point-clouds,” in *International Symposium on Robotics Research*, 2015.
- [2] 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.
- [3] M. Muja and D. G. Lowe, “Scalable nearest neighbor algorithms for high dimensional data,” *PAMI*, 2014.
- [4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *NIPS*, 2012.
- [5] R. Socher, J. Pennington, E. Huang, A. Ng, and C. Manning, “Semi-supervised recursive autoencoders for predicting sentiment distributions,” in *EMNLP*, 2011.
- [6] 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 *IROS*, 2008.
- [7] N. Srivastava and R. R. Salakhutdinov, “Multimodal learning with deep boltzmann machines,” in *NIPS*, 2012.
- [8] G. Hinton and R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *Science*, vol. 313, no. 5786, pp. 504–507, 2006.
- [9] I. Goodfellow, Q. Le, A. Saxe, H. Lee, and A. Y. Ng, “Measuring invariances in deep networks,” in *NIPS*, 2009.
- [10] K. Q. Weinberger, J. Blitzer, and L. K. Saul, “Distance metric learning for large margin nearest neighbor classification,” in *NIPS*, 2005.
- [11] J. Weston, S. Bengio, and N. Usunier, “Wsabie: Scaling up to large vocabulary image annotation,” in *IJCAI*, 2011.
- [12] J. Moore, S. Chen, T. Joachims, and D. Turnbull, “Learning to embed songs and tags for playlist prediction,” in *ISMIR*, 2012.
- [13] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, “Multimodal deep learning,” in *ICML*, 2011.
- [14] K. Sohn, W. Shang, and H. Lee, “Improved multimodal deep learning with variation of information,” in *NIPS*, 2014.
- [15] T. Mikolov, Q. V. Le, and I. Sutskever, “Exploiting similarities among languages for machine translation,” *CoRR*, 2013.
- [16] J. Hu, J. Lu, and Y.-P. Tan, “Discriminative deep metric learning for face verification in the wild,” in *CVPR*, 2014.
- [17] S. Miller, J. Van Den Berg, M. Fritz, T. Darrell, K. Goldberg, and P. Abbeel, “A geometric approach to robotic laundry folding,” *IJRR*, 2012.
- [18] M. Bollini, J. Barry, and D. Rus, “Bakebot: Baking cookies with the pr2,” in *IROS PR2 Workshop*, 2011.
- [19] M. Koval, N. Pollard, and S. Srinivasa, “Pre- and post-contact policy decomposition for planar contact manipulation under uncertainty,” *International Journal of Robotics Research*, August 2015.
- [20] J. Sung, B. Selman, and A. Saxena, “Synthesizing manipulation sequences for under-specified tasks using unrolled markov random fields,” in *IROS*, 2014.
- [21] D. Misra, J. Sung, K. Lee, and A. Saxena, “Tell me dave: Context-sensitive grounding of natural language to mobile manipulation instructions,” in *RSS*, 2014.
- [22] D. Kappler, P. Pastor, M. Kalakrishnan, M. Wuthrich, and S. Schaal, “Data-driven online decision making for autonomous manipulation,” in *R:SS*, 2015.
- [23] O. Kroemer, E. Ugur, E. Oztop, and J. Peters, “A kernel-based approach to direct action perception,” in *ICRA*, 2012.
- [24] D. Kappler, J. Bohg, and S. Schaal, “Leveraging big data for grasp planning,” in *ICRA*, 2015.
- [25] I. Lenz, H. Lee, and A. Saxena, “Deep learning for detecting robotic grasps,” *RSS*, 2013.
- [26] S. Levine, N. Wagener, and P. Abbeel, “Learning contact-rich manipulation skills with guided policy search,” *ICRA*, 2015.
- [27] I. Lenz, R. Knepper, and A. Saxena, “Deepmpc: Learning deep latent features for model predictive control,” in *RSS*, 2015.
- [28] I. Tsochantaris, T. Joachims, T. Hofmann, Y. Altun, and Y. Singer, “Large margin methods for structured and interdependent output variables,” *JMLR*, vol. 6, no. 9, 2005.
- [29] M. D. Zeiler, “Adadelta: An adaptive learning rate method,” *arXiv preprint arXiv:1212.5701*, 2012.
- [30] P. Vincent, H. Larochelle, Y. Bengio, et al., “Extracting and composing robust features with denoising autoencoders,” in *ICML*, 2008.
- [31] M. D. Zeiler, M. Ranzato, R. Monga, et al., “On rectified linear units for speech processing,” in *ICASSP*, 2013.
- [32] Y. Ioannou, B. Taati, R. Harrap, et al., “Difference of normals as a multi-scale operator in unorganized point clouds,” in *3DIMPVT*, 2012.
- [33] R. B. Rusu, “Semantic 3d object maps for everyday manipulation in human living environments,” *KI-Künstliche Intelligenz*, 2010.
- [34] M. Quigley, K. Conley, B. Gerkey, et al., “Ros: an open-source robot operating system,” in *ICRA workshop on open source software*, 2009.
- [35] F. Bastien, P. Lamblin, R. Pascanu, et al., “Theano: new features and speed improvements,” *Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop*, 2012.
- [36] L. Van der Maaten and G. Hinton, “Visualizing data using t-sne,” *JMLR*, vol. 9, no. 2579-2605, p. 85, 2008.