Learning Latent Representations of Environments and Grasping Motions for Fast Database Retrieval of Collision-Free Motions

○Solvi Arnold (Shinshu University), Keisuke Takeshita (Toyota Motor Corp.),
Takashi Yamamoto (Toyota Motor Corp.), Rui Zhu (Shinshu University),
Kotaro Nagahama (Shinshu University), and Kimitoshi Yamazaki (Shinshu University)

1. Introduction

Grasping and transporting objects such as cups and bottles are common aspects of household support. Humans perform these actions with little to no conscious cognitive effort, and would expect the same from household support robots. However, computing non-colliding grasp trajectories can be computationally challenging, particularly in obstacle-rich environments. This issue is further compounded by limitations in computation power imposed by affordability considerations. Consequently, it makes sense to make smart use of collections of precomputed grasping trajectories. Such strategies require a mechanism for quickly retrieving suitable candidate trajectories for the current environment from a large set of pre-computed trajectories. In this work we address this challenge, by proposing a strategy for learning representations of grasping motions and environments that facilitate quick database retrieval of candidate motions for a given environment, using small-scale neural networks.

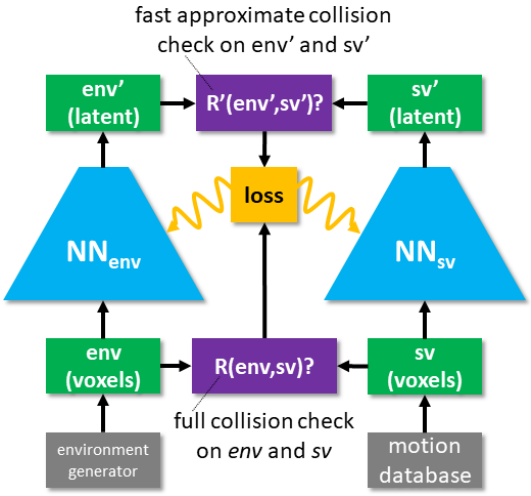

Fig. 1. System outline. *env* = environment, *sv* = motion swept volume, *NN* = neural network. *R* and *R'* denote collision checks in voxel and latent space, respectively.

2. Approach

The concept of using compact latent representations (LRs) of various types of data for search has broad applicability in fields such as reverse image search [1][2] and recommender systems [3], as well as for object detection on basis of a given target [4]. For image data, autoencoders are a common choice for learning the LR. If the LR effectively captures the defining characteristics of the data, then similar data will produce similar LRs. We can then use distance measures in latent space as a means for finding data items similar to a given *query* data item.

Here we attempt to extend this concept to the problem setting of retrieving suitable grasp motions from a database, given the current environment. This problem setting differs from similarity-based search in that the query item (an environment) and the items to be retrieved (grasping motions) are different in kind. While there is a type of resemblance between environments and their non-colliding motions (the motion's voxel representation should be fully contained in the negation of the environment's voxel representation), the search criterion cannot be defined in terms of simple similarity. Therefor, instead of comparing LRs using a distance measure, we use an asymmetrical comparator over motion LRs and environment LRs to determine whether the former is a *match* w.r.t. the latter. The comparator is set in advance, after which we train encoders for motion and environment data with the training criterion that application of the comparator to the learned LRs should produce True if the motion is collision-free w.r.t. the environment and False otherwise. We have found no examples of this problem setting in the literature, so our first goal here is to assess whether this strategy is viable.

In practice, the LRs would be used as follows. As part of the preparation of the robot's software, we generate a database of grasping motions for a wide variety of scenarios. Each motion is processed using the trained motion encoding neural network, and the resulting LR is stored along with the motion, as the motion's database key. During operation, when a grasping motion is required, a representation of the current environment is processed using the environment-encoding network to obtain the environment LR. This LR is then used as a search query to quickly retrieve a set of candidate motions from the database. Note that at run time, we only need to perform one forward propagation through a comparatively small neural network to obtain the search key, so computational cost is small.

3. Task

The task we adopt is grasping objects (cups, bottles) on a table, with a variable number of obstacle objects present

on the table. The robot platform is the Human Support Robot (HSR, Toyota Motor Corp.) [5].

We let both environments and grasping motions be represented as voxel volumes. We represent a motion as the voxel representation of its swept volume (SV), the set of points occupied by the robot at any point in time over the course of the motion.

For training data we use the dataset of [6], which contains 1000 examples of grasping motions performed by the HSR, along with their voxelised SVs. This set contains forward grasping motions and sideways grasping motions (500 examples each). For detailed description of the motion repertoire and data acquisition procedure we refer to [6]. Motions SVs are voxelised at a resolution of 34x28x24 (22848 voxels total). For motions, a voxels having value 1 indicates that the motion passes through that voxel.

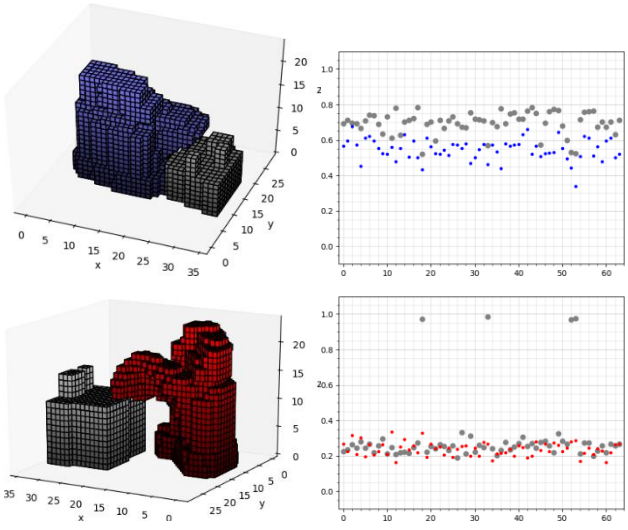

Fig. 2. Voxel representations and latent representations of environments (grey) and motion SVs (blue if safe, red if colliding). These LRs render R' True for the safe example and False for the colliding example, as intended.

Environments are not included in our dataset, but voxelisations of possible environments are easy and cheap to generate in large quantities. Our task setting defines the environment as a table with a target object and some number of obstacle objects on it. For convenience we let environment voxelisations have the same resolution (in practice one would crop the environment to the size of the motion SV), and use one voxel as our size unit below. We generate the environment as follows. We place a table centred on coordinate (29,14) in the XY plane. Table width and length are independently picked from the range [12,16], and height from the range [5,12]. The table is rotated around its z-axis by a random angle from range [0,90]. Next we place obstacle objects on the table surface. The number of obstacles is randomly selected from the range [0,3]. Obstacle sizes are picked (independently for each axis) from [2,4]. Positions are picked from the set of unoccupied voxels representing the table surface. For environments, voxel value 1 indicates that the voxel overlaps with an object (obstacle or table). The parameters of the environment generation routine are set so as to produce an approximately 50/50 split of collision/non-collision cases with our motion data, in order to minimise training bias.

The target object is assumed to be at the end position of the grasping motion, as this is the scenario we would want to check in practice. Contact with the target object is intentional and should not be counted as a collision, so we do not represent the target object in the environment voxelisation. Figure 2 shows examples of environments and SVs.

To ensure maximal environment variation during training, we perform environment generation as part of the training batch generation procedure. Consequently there is no fixed dataset of environments.

4. Network Training

Let *R(env,sv)* be a function that for a given environment *env* and motion swept volume *sv* determines whether the latter is collision free w.r.t. the former.

For each type of data, we construct an encoder network. The encoder networks share the same architecture, consisting of two fully-connected connection layers (we also experimented with 3D convolution layers, but obtained better results with the fully connected architecture). The hidden neuron layer has 4096 neurons, and the output layer has 64. All layers use the tanh activation function. This architecture produces latent representations consisting of 64 real values in the [-1,1] range.

We define a function *R'* as the latent-space counterpart of *R*, i.e. *R'* determines whether *env* and *sv* are collision free based on their latent representations *env'* and *sv'*:

$$R'(env', sv') = \bigwedge_{i=0}^{d} sv'_i \leq env'_i$$

Where *d* is the LR dimensionality (64 here) and *i* indexes the LR's elements. Note that *R'(env', sv') iff R(env, sv)* for many trivial encodings, e.g. the encoding that simply lists the voxel values and bit-flips *sv'*. Our learning problem is to find more compact encodings that still make *R'(env', sv') iff R(env, sv)* true for as many cases as possible.

The definition of *R'* is chosen to be suitable for database search. With this *R'*, each *env* effectively carves out a *d*-dimensional hyperrectangle in the latent motion space, which facilitates retrieval strategies such as tree search. Also note that this *R'* allows comparison against a given *sv'* to terminate the moment an sv'_i value for which $sv'_i > env'_i$ is encountered.

We define the training loss as follows:

$$loss(env', sv') = \begin{cases} loss^{fn}(env', sv') & if\ R(env, sv) \\ loss^{fp}(env', sv') & otherwise \end{cases}$$

$$loss^{fn}(env', sv') = \sum_{i=0}^{d} max(0, sv'_i - env'_i + \beta)$$

$$loss^{fp}(env', sv') = \sum_{i=0}^{d} m_i * max(0, env'_i - sv'_i + \beta)$$

Here β is a small positive margin to improve stability (set to 0.05 here). Sub-losses $loss^{fn}$ and $loss^{fp}$ are intended to resolve false negatives and false positives, respectively. Vector *m* is a random binary masking vector of length *d*, with each element having a 1/8 probability of taking value 1. We regenerate *m* for every evaluation of the loss function. Turning a false positive into a true negative requires that at least one element of *sv'* is made to exceed the corresponding element of *env'*, but it does not matter which element. We have no computationally cheap way of finding out which elements can be pushed around without negatively affecting the LRs of other examples, so (using *m*) we push a random selection, assuming that values for selected elements that have wiggling room will retain the modification while other elements will revert it in subsequent updates.

We train the nets for 20000 iterations using the signSGD update rule [7], with a learning rate (weight update step) of 10^{-6} and a batch size of 512.

We evaluate performance by pairing each motion from the database with 512 randomly generated environments, and comparing the outcomes of *R* and *R'*.

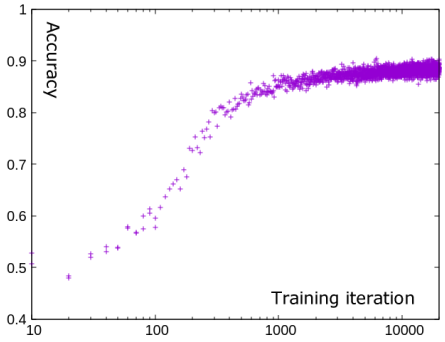

Fig. 3. Accuracy development over the training process. X axis (training iteration) in log scale.

5. Results

First we apply the evaluation routine on the untrained nets. Unsurprisingly, the encodings produced pre-training result in *R'* returning False for all evaluation cases (the probability of obtaining $sv_i < env_i$ on 64 out of 64 elements *i* when the values are effectively random is exceedingly low). Consequently, training accuracy starts out at approximately 50%. Subsequently, we see accuracy quickly climb to about 85%, after which progress tapers off before stabilising at approximately 88%. Figure 3 shows the training process. Figure 2 shows examples of motion SVs and generated environments, along with their representations in latent space, for a collision-case and a non-collision case. Table 1 shows the proportions of frequencies of all four possible outcomes.

We currently assume that the motion database is known at training time, and that encountered environments will be novel. This setup mixes known and unknown data, and consequently the usual division into training data and test data does not apply. In practice it could be useful to allow for new motions to be added to the database post-training. In this case we would either have to perform additional training, or ensure that the initially learned encoding generalises well to unseen motions. We intend to address this case in future work.

	True ($R' = R$)	False ($R' \neq R$)
Positive (R = True)	0.427	0.041
Negative (R = False)	0.454	0.077

Table 1. Proportions of true/false positives/negatives obtained in post-training evaluation.

6. Conclusions & Future Work

We proposed a strategy for learning suitable LRs for the motion retrieval problem. Our results suggest that this type of approach is viable. Future work includes refinement of the training algorithm to improve accuracy, extension to larger datasets and voxel resolutions, integration with a database search algorithm, and accommodation of post-training addition of new motions to the database.

References

[1] Krizhevsky, A. & Hinton, G. E.: "Using very deep autoencoders for content-based image retrieval," ESANN2011, 2011.

[2] Sharma S., Umar I., Ospina L., Wong D. & Tizhoosh H. R.: "Stacked Autoencoders for Medical Image Search," ISVC 2016. Lecture Notes in Computer Science, vol 10072, 2016, doi: 10.1007/978-3-319-50835-1_5

[3] Zhang, G., Liu, Y. & Jin, X.: "A survey of autoencoder-based recommender systems," Front. Comput. Sci. 14, 430–450, 2020, doi: 10.1007/s11704-018-8052-6

[4] Arnold, S., Ohno, K., Hamada, R. & Yamazaki, K.: "An image recognition system aimed at search activities using cyber search and rescue dogs," Journal of Field Robotics, 36(4), 677 – 695, 2019, doi: 10.1002/rob.21848

[5] Yamamoto, T., Terada, K., Ochiai, A., Saito, F., Asahara, Y. & Murase, K.: "Development of the Research Platform of a Domestic Mobile Manipulator Utilized for International Competition and Field Test," IROS2018, 7675–7682, 2018.

[6] Zhu, R., Nagahama, K., Takeshita, K. & Yamazaki, K.: "A Method of Online Motion Generation Using Swept Volumes Collected in Advance," IEEE ICMA2019, 1842-1847, 2019.

[7] Bernstein, J., Wang, Y., Azizzadenesheli, K. & Anandkumar, A.: "SignSGD: Compressed Optimisation for Non-Convex Problems," arXiv:1802.04434, 2018.