Learning from Demonstration using a Curvature Regularized Variational Auto-Encoder (CurvVAE)

Travers Rhodes, Tapomayukh Bhattacharjee, and Daniel D. Lee

***Abstract*— Learning intricate manipulation skills from human demonstrations requires good sample efficiency. We introduce a novel learning algorithm, the Curvature-regularized Variational Auto-Encoder (CurvVAE), to achieve this goal. The CurvVAE is able to model the natural variations in human-demonstrated trajectory data without overfitting. It does so by regularizing the curvature of the learned manifold. To showcase our algorithm, our robot learns an interpretable model of the variation in how humans acquire soft, slippery banana slices with a fork. We evaluate our learned trajectories on a physical robot system, resulting in banana slice acquisition performance better than current state-of-the-art.**

I. INTRODUCTION

Learning new skills is challenging for robots. In cases where humans know how to perform a skill, learning from demonstration is a common approach. In learning from demonstration, a human demonstrates some skill one or more times and the robot learns from those human demonstrations. Learning from demonstration can speed up robot learning because the robot can take advantage of human expertise.

One of the challenges of learning from demonstration, however, is the difficulty in collecting expert data. Human demonstrations take time to collect, and therefore learning from demonstration usually has a small number of examples, compared to the tens of thousands of examples present in standard machine learning datasets. For intricate manipulation skills, the learning from demonstration algorithm must be able to learn a complicated model of the skill with many parameters. In order to be sample efficient, the algorithm must include some form of inductive bias or model regularization.

There are many forms of model regularization. In this work, we study a regularization scheme based on curvature. Functions that overfit the data will tend to have higher curvature, so we penalize the curvature of the learned model in our regularization term. We present a novel learning algorithm, the Curvature-regularized Variational Auto-Encoder (CurvVAE), which combines the modeling capabilities of a Variational Auto-Encoder (VAE) [1] with a curvature regularization term to prevent overfitting to small datasets.

To showcase learning from demonstration using a CurvVAE, we choose an example which requires intricate manipulations and is hard to simulate. The domain we choose is the acquisition of soft and slippery food items using a fork. In particular, our robot learns to pick up banana slices with a fork, as shown in Fig. 1. This problem requires intricate manipulations, since simple stabbing motions will often lead to the food slipping off the fork prongs. It is hard to simulate this problem because we lack good models of the deformable nature of banana slices and of the relevant frictional forces. Since humans can demonstrate how to acquire banana slices with a fork, this problem is a good candidate for learning from demonstration, as visualized in Fig. 2. However, since learning from demonstration on this problem requires real human examples, the training dataset is relatively small. Our CurvVAE algorithm is able to learn a good model of human trajectories in an interpretable way without overfitting.

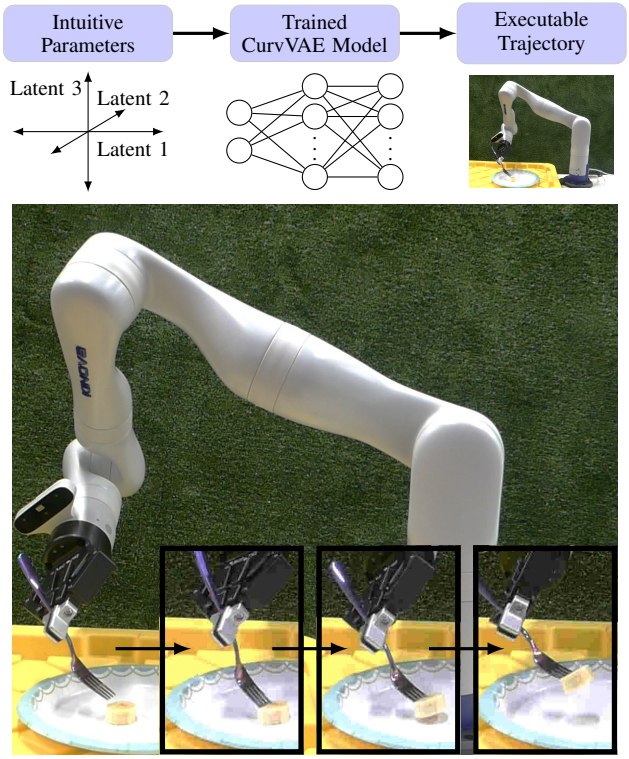

Fig. 1. Execution of a parameterized trajectory from a Curvature-regularized Variational Auto-Encoder (CurvVAE)

The contributions of this paper are:

- our novel CurvVAE algorithm to learn intricate manipulation skills from demonstrations, including the use of a curvature regularization term to prevent overfitting to small datasets, and
- the application of this algorithm to the manipulation of soft, spearable, slippery food (i.e. banana slices), including evaluation on a physical robotic system resulting in performance better than current state-of-the-art [2].

[1] All authors are with Cornell University, USA {tsr42,tb557,ddl46}@cornelledu

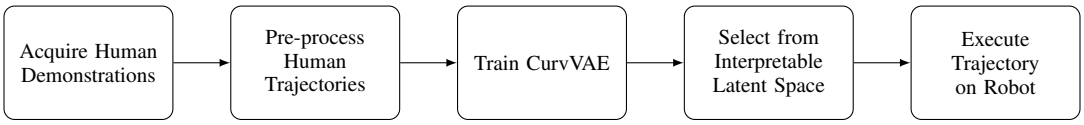

Fig. 2. Component diagram of how a CurvVAE model is used for learning from demonstration. A sample of human demonstrations is collected. The dataset is truncated and time-warped to provide an aligned dataset of human demonstrations. We train a CurvVAE to model the variations in how humans perform the skill, generating an interpretable, learned parameterization of the skill. The user selects their desired parameters from the interpretable latent set, and the CurvVAE can then generate a trajectory that is executed on the physical robotic arm.

II. RELATED WORK

A. Trajectory Representation

In the robotics context, trajectories are often represented by key points of splines [3], [4], [5] or by parameters of Dynamic Movement Primitives (DMPs) [5], [6], [7], [8]. While it is easy to time-warp and change the goal position of DMPs, the full set of parameters in both DMPs and splines is generally too large to be conveniently modified by hand for trajectories of robots with multiple joints. By contrast, our work learns an intuitive, very low-dimensional representation of trajectories with human interpretabilty in mind.

Our architecture is flexible. We could have used the parameters of splines or DMPs as the input to our CurvVAE. Such an approach would be similar to [6], which uses the DMP parameters as the input to Principal Components Analysis (PCA) to learn the natural variation of the data. CurvVAE discourages curvature but does not force exact linearity like PCA does, so our model is more flexible and makes fewer linearity assumptions than the approach presented in [6].

Some authors have trained task-specific representations, which are functions from task variables, like the height of a known obstacle or the location of a dartboard target, to DMP parameters [6], [8]. However, this type of task-parameterized DMP is a supervised problem where the desired representation, namely the task variable, is known a priori. Our work is based on the unsupervised learning of the representation of trajectories.

B. Curvature Regularization

Considering models with regularization terms related to curvature, we note important differences between our work and previous work. [9] regularize the Hessian (multivariate second derivative) and equate that regularization with the regularization of a function's "curvature profile." However, the extrinsic curvature of a manifold and the Hessian are not equivalent. Scaling the input space can significantly reduce the Hessian without affecting the extrinsic curvature of the output manifold. We want our curvature regularization term to penalize curvature of the learned manifold, not to encourage a particular scaling of the latent space. The authors are not aware of previous work which penalizes the extrinsic curvature of the learned manifold of a VAE.

III. CURVATURE REGULARIZED VAE

A. Variational Auto-Encoder (VAE)

We use machine learning to model the human demonstrations. We want to model the full range of natural variation in the demonstrations, not just the average demonstration, so we use an unsupervised modeling approach that can identify the main factors of natural variation in the data. A Variational Auto-Encoder (VAE) is an unsupervised machine learning approach that models the main factors of variation of the data [1]. The VAE includes an embedding function that takes in a datapoint and returns an associated lower-dimensional latent representation, corresponding to the main factors of variation of the data while removing noise. The VAE also includes a data generation function that takes in a latent value and generates an associated synthetic datapoint. The VAE is trained so that the data generation function creates synthetic data that looks like the training dataset and maintains the main factors of variation in the data. The VAE averages out noise in the data by using a cost that tries to make sure nearby points in the latent space generate similar synthetic data points.

Specifically, the VAE training loss contains a noisy reconstruction loss, which is computed by embedding a training point into the latent space, perturbing that latent embedding according to a noise distribution, reconstructing back from the latent space into the data space, and then computing the error between the reconstructed data point and original training point. The reconstruction loss encourages the VAE to act like an auto-encoder and learn an invertible encoding.

The VAE training loss also includes a KL-divergence loss on the noisy embedding distributions which encourages the embedding values to stay near the origin and discourages the embedding noise from collapsing to the zero noise, perfect auto-encoder limit. The β-VAE multiplies the KL-divergence term by a tunable hyperparameter β to trade off the KL-divergence objective with reconstruction accuracy [10]. The full β-VAE training loss is defined as the sum over the losses for each training data point x, where the loss for a single datapoint x, assuming a Gaussian noise model, is

$$\|x - g(f(x) + \epsilon)\|^2 + \beta \frac{\|f(x)\|^2 + \sum_d (\sigma_d^2(x) + \log(\sigma_d^2(x))}{2}$$

where g is the generator function, f is the embedding function, $\sigma_d^2(x)$ is the variance of our embedding noise in the d^{th} dimension, and ϵ is sampled from an axis-aligned, multivariate normal distribution with variance in each dimension defined by $\sigma_d^2(x)$. f, g, and σ^2 are all trainable neural networks.

The negative of this loss plus a constant is called the Evidence Lower Bound (ELBO), since it is a lower bound on $\log(p(x)) = \log(\int_z p(x|z)p(z)dz)$, which is the log-likelihood of the data point given our model, marginalized

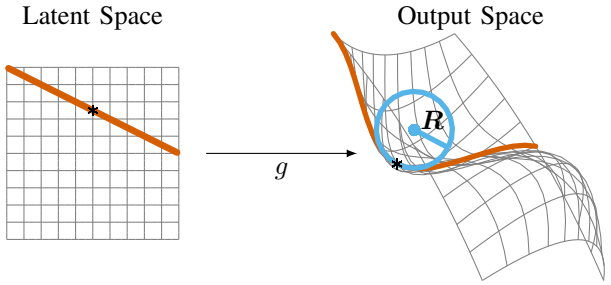

Fig. 3. After mapping a random point and line through g, we compute the radius R of the best-fit tangent circle to the resulting curve. Our regularization cost is $\frac{1}{R^2}$.

over all the possible latent values for that datapoint. Thus, maximizing the ELBO (minimizing our cost) is a proxy for maximum likelihood estimation.

B. Curvature-regularized VAE

VAEs and other neural-network-based machine learning algorithms generally require large datasets. In cases of limited data, models can benefit from regularization. Model regularization helps prevent overfitting the data, at the cost of adding inductive bias. However, inductive bias can have benefits, including making the generative model more human-understandable and disentangling latent factors of variation [11]. In this work we explore using curvature regularization to improve the sample efficiency of VAEs.

To regularize our model, we penalize the extrinsic curvature of the manifold learned by the generative function. Explicitly, we penalize the square of the reciprocal of the radius of curvature of curves in the learned manifold. The curves are selected by taking random lines through a point in the latent space and mapping them through the function g onto the curved output manifold embedded in the data space, as shown in Fig. 3. We average the associated extrinsic curvatures for different randomly sampled points and lines to get our curvature estimate of the overall manifold. Regularizing the curvature reduces the complexity of the learned model and encourages simpler functions. Additionally, this regularization term encourages interpretable latent variables in the model, since it explicitly causes the model to seek out a generator function for which linear changes to the latent space lead to closer-to-linear changes in the output.

For a generator function as g, we use finite differences to estimate the curvature penalty as:

$$\left(\frac{\frac{g(z+h)-g(z)}{\|g(z+h)-g(z)\|+\varepsilon} - \frac{g(z)-g(z-h)}{\|g(z)-g(z-h)\|+\varepsilon}}{\|g(z+h)-g(z)\|+\varepsilon} \right)^2$$

where z is randomly sampled from the latent space, h is randomly sampled from a spherical shell with small radius, and ε is a small constant (ε =1e-10) to avoid divide-by-zero errors. Since h is randomly sampled from a spherical shell, replacing h with $-h$ in the definition above leads to an equivalent estimate of curvature. This cost is scaled by a hyperparameter γ. We call the β-VAE architecture with this curvature regularization a Curvature-regularized Variational Auto-Encoder (CurvVAE). The full CurvVAE training loss is thus the sum over each training data point x of:

$$\|x - g(z)\|^2 + \beta \frac{\|f(x)\|^2 + \sum_d (\sigma_d^2(x) + \log(\sigma_d^2(x))}{2} + \gamma \left(\frac{\frac{g(z+h)-g(z)}{\|g(z+h)-g(z)\|+\varepsilon} - \frac{g(z)-g(z-h)}{\|g(z)-g(z-h)\|+\varepsilon}}{\|g(z+h)-g(z)\|+\varepsilon} \right)^2$$

where $z = f(x) + \epsilon$, h is randomly sampled from a spherical shell with small radius, g is the generator function, f is the embedding function, $\sigma_d^2(x)$ is the variance of our embedding noise in the d^{th} dimension, ϵ is sampled from an axis-aligned, multivariate normal distribution with variance in each dimension defined by $\sigma_d^2(x)$, and ε is a small constant to avoid divide-by-zero errors.

C. CurvVAE Example on Fork Poses

To visualize the effect of the curvature regularization term in the CurvVAE loss, we consider a sample dataset of fork poses. These poses are the combined 9,920 poses in the fork trajectories we describe in Section IV-A. We represent each fork pose using three spatial dimensions and four quaternion dimensions. Each fork pose is expressed as a vector of dimension 7, and we expect that the sampled fork poses can be well approximated by a lower-dimensional manifold reflecting the largest factors of variation in the pose dataset.

We then take a very small training dataset of two hundred randomly sampled poses and use the rest of the dataset as the test set. We train 13 different β values for β-VAE and 16 different γ values for CurvVAE (holding β at 1e-5). For each hyperparameter set, we train 10 different models at 3 different learning rates, for a total of 390 different β-VAE models and 480 different CurvVAE models. Our models all use 3 latent dimensions.

After training each model, we plot the reconstruction root mean squared error on the training set and on the test set. These results are shown in Fig. 4, along with the results for PCA models on zero through three dimensions. We note that no value of β for the β-VAE can achieve as low test error as the optimal CurvVAE. We additionally note that though CurvVAE regularizes to encourage a more linear model, it does not require a linear model, so our CurvVAE is also able to achieve a lower test error than PCA.

IV. MACHINE LEARNING METHODS

Since CurvVAE allows us to train a VAE on a relatively small dataset, we can use a CurvVAE for robotic learning from demonstration in cases where human demonstration data is limited. As an example, we learn a generative model of human fork trajectories acquiring food. We then show that the learned motions have interpretable latent spaces and lead to effective trajectories when run on a physical robot arm.

A. Human-demonstrated Trajectories and Data Processing

We use recordings of human participants picking up banana slices with a fork as our training demonstrations [12]. That fork trajectory dataset contains detailed fork poses (both

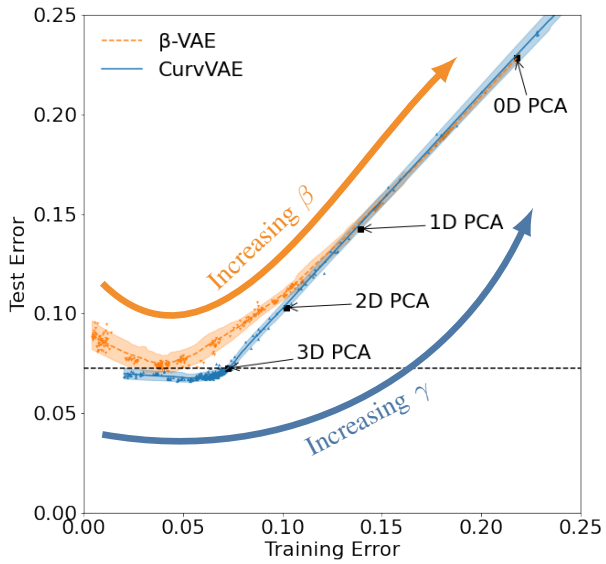

Fig. 4. β-VAE is not able to attain as low test error as our CurvVAE model. Our CurvVAE also outperforms PCA.

positions and orientations) for real food acquisition and delivery trajectories in a pretend assistive feeding environment. In the recordings, participants move a fork to a plate, pick up a banana slice with the fork, and then pretend to feed it to a mannequin.

For this work, we just want to model the part of the trajectory where the user is picking up the food, so we extract the individual banana slice acquisition components from the longer demonstration trajectories. However, since we want to learn a continuous model of the trajectory styles, we remove trajectories which are categorically different from others, including those with outlier fork orientations. For the same reason, we remove trajectory sequences from what appears to be a left-handed participant and we remove trajectory sequences where the force-torque sensor on the fork does not detect food impact. We truncate trajectories to start at most one centimeter above food impact and to end at most four centimeters above the plate, since we are just interested in modeling the manipulation strategy used to pick up the food and bring it to a stable fork position. After data preprocessing, the dataset has 155 banana acquisition examples.

We linearly scale the timing of each sequence so that each trajectory sequence starts at time zero and lasts one arbitrary time unit. Additionally, we translate each trajectory so that the maximal force recording from the trajectory occurs at the X/Y origin and the lowest height recorded during the trajectory occurs at Z=0. These two steps align trajectories, and we can translate or time-warp trajectories during replay. We mean-center the poses and then multiply all the positions by a scaling factor of 5.6 (computed to make the largest variance in the X, Y, or Z direction across the whole dataset equal to one) and multiply all the quaternion orientations by 0.90 (computed so that rotations around the fork tip that move the handle one centimeter are penalized roughly the same amount as translations which move the fork tip one centimeter). When the model is applied and poses are generated for the robot to execute, these centering and scaling transformations are inverted.

B. Neural Network Architecture for Trajectories

Our trajectory generator function g takes as input a three-dimensional latent value and an additional time parameter. It outputs a pose for that trajectory at that associated time. The pose output by the model is represented as a vector of length seven (three position values and four quaternion values). The generator is a neural network with a single hidden layer of a thousand nodes and uses rectified linear units as its nonlinearities. A shallow but wide architecture is a simple architecture for learning continuous functions. By sweeping the time parameter between zero and one, a full pose trajectory for the fork can be generated from this architecture.

The embedding function f has a shared hidden layer with a thousand nodes and two linear heads attached, one to predict the mean embedding z, and one to predict the log variance of the embedding $\ln(\sigma^2)$, again using rectified linear units for nonlinearities. We follow the standard assumption of axis-aligned Gaussian embeddings, so that for a three-dimensional embedding latent space, z and $\ln(\sigma^2)$ are both three-dimensional vectors.

Since our data points are trajectories, we have a bit of additional infrastructure in our training architecture. To train our reconstruction loss, we randomly pull two poses and their associated timestamps (x_a, t_a) and (x_b, t_b) from a single training trajectory. x_a and x_b are seven-dimensional (position and quaternion) vectors; t_a and t_b are their respective scalar timestamps. We embed (x_a, t_a) using our embedding function f to get a three-dimensional latent embedding z_a and a log-variance estimate $\ln(\sigma_a^2)$. We add Gaussian noise to the mean estimate following the distribution defined by $\sigma_a^2 = \exp(\ln(\sigma_a^2))$ to get the noisy embedding $\tilde{z}_a$. This noisy embedding should be an approximate latent representation of the entire trajectory associated with (x_a, t_a), from which (x_b, t_b) was also drawn. We now pair this embedding with timestamp t_b, and use the generator function g to compute a reconstructed pose for $(\tilde{z}_a, t_b)$ with the intent that it be a good estimate of x_b. The squared error between $g(\tilde{z}_a, t_b)$ and x_b is our reconstruction loss.

Even though our generator function g is now additionally parameterized by time, we only apply the curvature loss calculations to perturbations of the latent value, keeping the randomly-sampled time parameter constant within each curvature loss calculation. That is, we do not penalize the curvature of the individual trajectories over time, we only penalize how trajectories change as we vary the latent value.

C. Training Hyperparameters

We trained a CurvVAE model using the above model architecture on training data of forks picking up banana slices. We chose hyperparameters based on the tradeoff between latent space interpretability and reconstruction accuracy. Lower

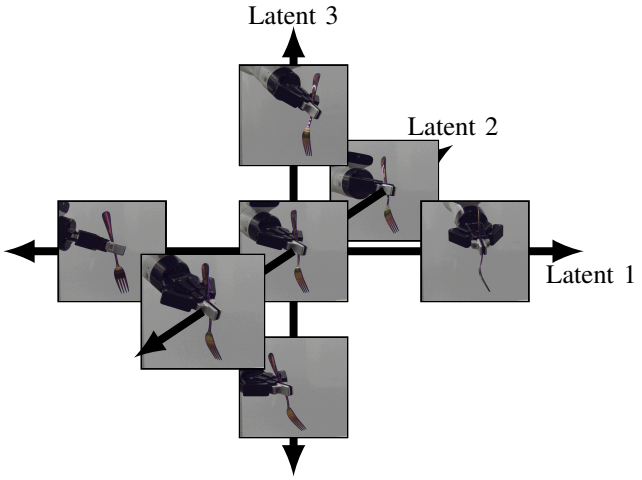

Fig. 5. Fork starting poses for the CurvVAE trajectory. Each latent direction rotates the fork along a different axis.

β ensures good reconstruction accuracy, but too small a β winds up embedding the latent points very far from each other in the latent space. For our training dataset, $\beta = 0.001$ was a good compromise between these goals. Increasing γ leads to latent traversals that are closer to linear in the output space, at the cost of lower reconstruction accuracy. For our training dataset, $\gamma = 0.001$ led to interpretable models that still had good reconstruction accuracy. We train for 3,000 batches of 256 pairs of poses randomly sampled from trajectories.

D. Model Interpretability

The CurvVAE learns an intuitive latent space of trajectories. Changing the latent value leads to interpretable changes to the starting pose of the fork, corresponding to rotations of the fork around different axes. Fig. 5 shows the change in fork starting pose as we vary the three-dimensional latent value along different dimensions. The first latent value corresponds to rotation of the starting pose of the fork around the vertical axis. The second latent value corresponds to tipping the handle away/toward the viewer. The third latent value corresponds to tipping the handle to the left/right.

V. PHYSICAL ROBOT EXPERIMENTS

A. The Robot Arm

Experiments are performed on a 7-DOF Kinova Gen3 Arm [13] with a two-fingered Robotiq gripper (85mm stroke width) [14]. A metal fork is held at a calibrated location and orientation in the gripper. We run a 500Hz control loop on an Intel Core i7 CPU running a realtime version of Ubuntu to send effort commands to the arm. The control computer converts high-level desired joint-position commands to low-level effort commands to send to the arm using PID control.

B. Trajectory-following Implementation

The target trajectory is encoded as 64 waypoint poses to be executed in 6.4 seconds. We use a Jacobian-based controller to estimate the joint changes required to move the fork-tip held in the robot end-effector from a starting pose through the desired waypoints. These joint changes are scaled as necessary to ensure the robot does not move unreasonably quickly, and together form a set of desired joint angle waypoints. These desired joint angles are sent to the control computer alongside a desired time-to-goal for each value of desired joint angles. The realtime control computer computes intermediate waypoint joint positions to move each joint linearly from its current actual angle to arrive at the next desired angle at the desired time. PID control is then used to send computed effort commands to the robot arm at 500Hz to follow the intermediate waypoint joint positions.

C. CurvVAE Latent Value Selection

The latent values of the trained CurvVAE are interpretable, as seen in Fig. 5, and correspond primarily to different starting orientations of the fork. We choose latent values that approach the banana from approximately the same direction and that ensure that no fork prong is tipped closer to the table than the other prongs. Under those constraints, we then find latent values that tilt the fork away from the banana slice at various angles. In particular, we choose latent values that tilt the fork so that the prongs start 30° (latent values (0,-1.2,0.1)), 45° (latent values (0,-2.3,-1.5)), and 60° (latent values (0,-3.4,-3.1)) off of vertical. These trajectories are visualized in Fig. 6.

D. Baseline Trajectories

We compare the food acquisition efficiency of these learned CurvVAE trajectories to baseline trajectories inspired by [2]. The state-of-the-art trajectory presented in that work holds the fork handle at a 45° angle and moves the fork along a 45° line to skewer the food. The fork prongs are at a 75° angle during motion, so we label this trajectory the 75° baseline. We additionally test holding the fork prongs at a 45° angle while moving them at a 45° angle to skewer the banana slice. For both baseline trajectories, after the banana is skewered, the fork attempts to lift it vertically (without changing the orientation of the prongs). We choose these as our baselines because they are easy to code, intuitive, and include the current state of the art of [2].

We also include a β-VAE model trained on the same dataset and model architecture ($\beta = 0.001$) and a PCA model, both with the same latent dimension of three. We again choose latent values for these models that give similar starting angles to those chosen for the CurvVAE model.

E. Data Collection

We test food acquisition success on banana slices in a controlled laboratory setting. Bananas are sliced to be $\sim$1.5cm thick and placed on a marked location on a plate in a known position relative to the robot base. For each banana slice, we test multiple strategies. By testing multiple different strategies on the each slice in a randomized order, we control for variations in banana slices. We test each strategy 25 different times and count the number of times that the food was successfully acquired, where success requires staying

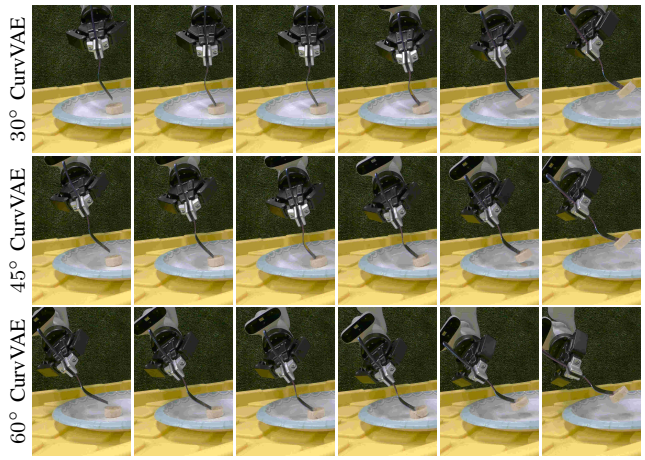

Fig. 6. Still images from CurvVAE trajectories. Trajectories were generated from the same model using different latent values.

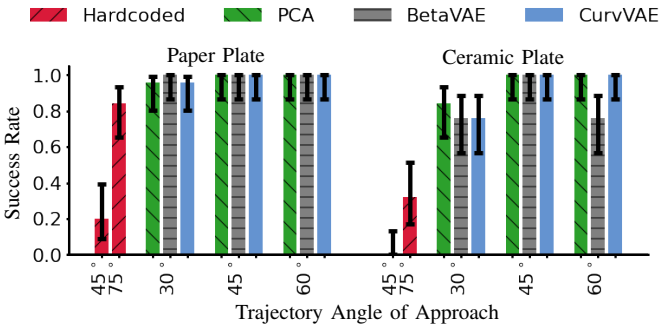

Fig. 7. Banana slice acquisition success rates.

on the fork for at least five seconds. Since the type of plate can affect food acquisition performance, we run the entire experiment twice, once using a paper plate and once using a ceramic plate. The ceramic plate has less surface friction, and therefore makes for a more challenging environment.

VI. RESULTS AND DISCUSSION

The results of the food efficiency acquisition experiments are presented in Fig. 7. Like [2], we see that soft food acquisition can be challenging for the hardcoded trajectories because banana slices are slippery and can slide off the prongs of the fork during lifting. However, we note that the trajectories learned from demonstration are consistently better at picking up banana slices. The difference is not statistically significant when the experiment is conducted on a paper plate. However, when the experiment is conducted on a ceramic plate, which has less friction and allows the banana to slide more easily, the performance increase is much larger and is statistically significant ($p < 0.05$).

While this manipulation task suggests that CurvVAE may outperform β-VAE, in general this manipulation task is not sensitive enough to differentiate CurvVAE performance from the other learning from demonstration strategies.

In this work, we showed how robots can learn an intuitive latent space from small samples using a CurvVAE. We applied the CurvVAE to learning from demonstration and showed that the learned latent space is low-dimensional enough that it can be meaningfully understood and could therefore be directly controlled. We further experimentally validated that the generated trajectories from the CurvVAE model have a success rate better than hardcoded baseline trajectories, including the state-of-the-art 75° baseline presented in [2]. In future work, we plan to incorporate feedback from the user during trajectory execution, and use the low-dimensional latent space learned by the CurvVAE for downstream learning. Additionally, we plan to extend this work to more types of food and more complicated feeding trajectories.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their helpful suggestions. We gratefully acknowledge support from National Science Foundation IIS grant #2132846.

REFERENCES

[1] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2014.
[2] R. Feng, Y. Kim, G. Lee, E. K. Gordon, M. Schmittle, S. Kumar, T. Bhattacharjee, and S. S. Srinivasa, "Robot-assisted feeding: Generalizing skewering strategies across food items on a realistic plate," *arXiv preprint arXiv:1906.02350*, 2019.
[3] H. Miyamoto, F. Gandolfo, H. Gomi, S. Schaal, Y. Koike, R. Osu, E. Nakano, Y. Wada, and M. Kawato, "Kendama learning robot based on a dynamic optimization theory," in *Robot and Human Communication - Proceedings of the IEEE International Workshop*, 1995, pp. 327–332.
[4] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *ICML*, vol. 97, 1997, pp. 12–20.
[5] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural Networks*, vol. 21, pp. 682–697, 2008.
[6] T. Matsubara, S. H. Hyon, and J. Morimoto, "Learning parametric dynamic movement primitives from multiple demonstrations," *Neural Networks*, vol. 24, no. 5, pp. 493–500, 2011.
[7] A. Pervez and D. Lee, "Learning task-parameterized dynamic movement primitives using mixture of GMMs," *Intelligent Service Robotics*, vol. 11, no. 1, pp. 61–78, jan 2018.
[8] B. C. Da Silva, G. Konidaris, and A. G. Barto, "Learning parameterized skills," in *ICML*, vol. 2, 2012, pp. 1679–1686.
[9] S. M. Moosavi-Dezfooli, A. Fawzi, J. Uesato, and P. Frossard, "Robustness via curvature regularization, and vice versa," in *CVPR*, vol. 2019-June, 2019, pp. 9070–9078.
[10] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "β-VAE: Learning basic visual concepts with a constrained variational framework," in *ICLR*, 2017.
[11] F. Locatello, S. Bauer, M. Lucic, G. Raetsch, S. Gelly, B. Schölkopf, and O. Bachem, "Challenging common assumptions in the unsupervised learning of disentangled representations," in *ICML*. PMLR, 2019, pp. 4114–4124.
[12] T. Bhattacharjee, H. Song, G. Lee, and S. S. Srinivasa, "A Dataset of Food Manipulation Strategies," 2018. [Online]. Available: https://doi.org/10.7910/DVN/8TTXZ7
[13] Kinova, "Gen3 robots," 2021. [Online]. Available: https://www.kinovarobotics.com/product/gen3-robots
[14] Robotiq, "2f-85 gripper," 2021. [Online]. Available: https://robotiq.com/products/2f85-140-adaptive-robot-gripper