Neural Networks for Inverse Kinematics Problems in Robotics

Franziska Schwaiger
Matriculation number: 03658670

Thomas Barthel Brunner
Matriculation number: 03675118

Introduction

In our project, we are evaluating the feasibility of using neural networks in robotics. More specifically, we are investigating the performance of Invertible Neural Networks (INN) and Conditional Variational Autoencoders (cVAE) for learning the inverse kinematics of a planar robotic arm with revoulte joints. In this report, we outline the progress made and discuss the results we have obtained.

Methods

Robot Simulations

To evaluate the performance of the architectures, we developed simulations of planar robotic arms with revolute joints for an arbitraty number of degrees of freedom (DoFs). In general, the forward kinematics equations of a planar robot arm with link lengths l_i and N revolute joints with joint angles $\theta_i \in [-\pi, \pi]$ can be described as:

$$x_{\text{TCP}} = \sum_{i=1}^{N} l_i \cos\left(\sum_{j=1}^{i} \theta_j\right), \; y_{\text{TCP}} = \sum_{i=1}^{N} l_i \sin\left(\sum_{j=1}^{i} \theta_j\right) \tag{1}$$

To improve the training and testing performance of our models, we modified these equations for fast vectorized numerical computation. The resulting operations and be seen in Equation 2, with joint angles vector $\boldsymbol{\theta} = [\theta_1, \theta_2, ...]^{\text{T}}$, the upper triangular matrix with ones as elements $\boldsymbol{U}$ and the link lengths vector $\boldsymbol{l} = [l_1, l_2, ...]^{\text{T}}$.

$$x_{\text{TCP}} = \cos\left(\boldsymbol{\theta}^{\text{T}}\, \boldsymbol{U}\right) \boldsymbol{l}, \; y_{\text{TCP}} = \sin\left(\boldsymbol{\theta}^{\text{T}}\, \boldsymbol{U}\right) \boldsymbol{l} \tag{2}$$

In our current setup, we do not take the angle of the tool center point (TCP) into consideration. Thus, the location of the TCP is defined by its x, y coordinates. As a result of this, all simulations with more than 2 DoF can have up to infinite solutions of the inverse kinematics for a given TCP position. The 2 DoF robot arm can have up to two solutions.

Dataset

We generated four different datasets for different robot configurations. Each dataset is composed of 10^6 samples of robot configurations and corresponding TCP coordinates. In our experiments, we used simulations with 4, 6, 8 and 10 DoF. Each configuration was sampled from a normal distribution $\theta_i \sim \mathcal{N}(\mu = 0, \sigma = 0.2)$. Thus, the dataset is composed mostly of configurations in which the robot arm is extended. This reflects real-world use cases of robot arms, which have limited workspaces and whose tasks are focused on one section of the workspace. Moreover, limiting the range of the joints improved the performance of the networks, as it simplifies the problem and as it avoids the discontinuity in the angles at $\theta = \pm\pi$. Figure 1 shows an illustration of the datasets.

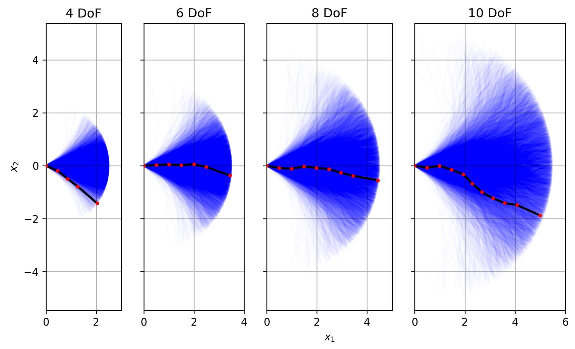


Fig. 1. Illustration of a subset of the datasets (10^4 samples) of the robot configurations.

Considering Singularities

As was already mentioned in the previous section, we avoid discontinuities by restricting the configuration space of the robot. Yet we implemented two approaches which consider this discontinuity in the angles at $\theta_i = \pm\pi$ to extend the implemented networks to inverse kinematics problems with general configuration spaces.

The first approach is a non-minimal parameterization of the configuration space that uses an augmented vector-based representation of the angles, which was already described in the midterm report. The drawback of this approach is its larger input space, which makes the task increasingly harder to solve for robots with higher DoFs.

For this reason, we also considered an alternative approach, in which we the discontinuities are considered in the loss function of the reconstruction error. It ensures that the computation of distances between angles takes the orientation of these angles into consideration. Thus, it avoids large negative losses for angles with distinct values, but whose orientation is similar.

Network Architectures

As described in the midterm report, we implemented and investigated two network architectures for generating the full posterior distribution of the joint angles: conditional Variational Autoencoders and Invertible Neural Networks.

Unlike Variational Autoencoders (VAE) [1], conditional Variational Autoencoders (cVAE) [2] are able to control the data generation process by conditioning the latent space z. Considering the application of inverse kinematics, samples are drawn from $p(z) \sim N(0, 1)$ and the predicted posterior distribution of the joint angles is then generated conditioned on the end-effector position.

Invertible Neural Networks (INN) [3] model a bijective mapping from the joint angles to the end-effector position by stacking invertible blocks together. Here, the dimensionality of the joint angles x is the same as the concatenated output consisting of the latent space z and the end-effector position y. The predicted posterior of the joint angles is then generated similar to cVAE by sampling from $p(z) \sim N(0, 1)$ and then running the network backwards conditioned on y.

Additionally, INNs share properties with normalizing flows [4], [5] which gradually transform a normal density into the desired data density. They are also bijective and for both normalizing flows and INNs, a tractable Jacobian exists which allows explicit computation of posterior probabilities. As the focus in this work lies on the comparison between cVAE and INN, normalizing flows are not considered anymore.

As stated in [3], INNs only need to be trained on the well-understood forward process. The inverse process can be obtained for free by just running the network backwards at prediction time. The authors also mention that results can be improved with additional unsupervised backward training. When applying INNs to the inverse kinematics problem, we found that unsupervised backward training is crucial to match the performance of the cVAE. This results in two network passes per training step to accumulate the gradients, which leads to much slower training. Additionally, when training INNs the parameters need to be adjusted carefully as training can become unstable.

Hyperparameter Optimization

As stated before, we trained a network of each architecture (cVAE and INN) for each dataset (4, 6, 8 and 10 DoF), which resulted in 8 models with different complexities. The sizes and hyperparameters of each model had to match the complexity of the simulation. To compare the performance of the networks across different DoFs in a fair setting, we used Tune [6] and Scikit Optimize [7] to perform a black-box optimization of the hyperparameters. For each model, 100 sets of hyperparameters were sampled, which were then used to train the networks using a subset of the main dataset (10^4 samples). We opted for a reduced number of samples to help reduce the computation time of the hyperparameter search.

For the cVAE, we optimized the learning rate, the weight decay, the number of hidden layers for the encoder and decoder and the amount of neurons per layer. For the INN, we optimized the learning rate, the weight decay, the number of coupling layers, the amount of layers per subnet and also the number of neurons per subnet layer. The ranges of hyperparameters used can be seen in Tables I and II. These values were based on intuition and previous experience in training these networks.

Implementation

The previous two model architectures (cVAE and INN) have been implemented in PyTorch and are inspired by existing implementations in [8], [9]. We used the Google Compute Engine to perform the hyperparameter search and to train our models.

Experimental Evaluation

Evaluation protocol

We trained each model using the best performing hyperparameters from the hyperparameter search. In total, we have four INN and four cVAE models, each corresponding to a different simulation (4, 6, 8 or 10 DoF). The batch size used during training was 1000 for both network architectures. All models were trained for 60 epochs on a dataset with 1 million samples and a train and test split of 70% and 30%, respectively. For evaluating the performance of each model, we used two evaluation metrics which have already been described in the midterm report and in [10]:

1. The average posterior mismatch between the distribution generated by the model and the ground truth estimate obtained via rejection sampling. For generating the posterior, we used 100 samples per ground truth position and averaged over 1000 samples from the test dataset.

2. The average re-simulation error as the mean squared distance between the ground truth end-effector position and the re-simulated end-effector position obtained from the predicted joint angles. The re-simulation error was averaged over the whole test dataset.

Results

The best performing hyperparameters from the hyperparameter search for the cVAE and INN are shown in Tables III and IV, respectively. Due to the hyperparameter search, the number of parameters of the models differ by a large amount. As it can be seen in the tables, the best performing models are not the ones with the greatest number of trainable parameters. This serves to show that the hyperparameters search space is sufficient.

The results of the evaluations (the mismatch of the posterior distribution and the re-simulation error) are plotted over the number of parameters in Figures 2 and 3, respectively. Each plotted point is labelled with the corresponding DoF. From Figure 2 we can see that the mismatch of the predicted posterior with the ground truth posterior obtained from rejection sampling is smaller for the INN when compared with the cVAE. However, as seen in Figure 3, the cVAE performs consistently better on the re-simulation error when compared with the INN.

TABLE I
RANGES OF INN HYPERPARAMETERS USED DURING HYPERPARAMETER SEARCH.

	Learning rate	# Subnet layers	# Coupling layers	# Neurons per layer	Weight decay
Min	0.0001	3	6	100	0.00001
Max	0.005	7	10	300	0.001

TABLE II
RANGES OF CVAE HYPERPARAMETERS USED DURING HYPERPARAMETER SEARCH.

	Learning rate	# Layers	# Neurons per layer	Weight decay
Min	0.0001	3	200	0.00001
Max	0.01	15	500	0.001

TABLE III
BEST-PERFORMING HYPERPARAMETERS FOR CVAE TRAINED ON A PLANAR ROBOT WITH REVOLUTE JOINTS

DoF	Learning rate	# Layers	# Neurons per layer	Weight decay	# Trainable parameters
4	0.009	3	230	0.0002	217128
6	0.0041	3	200	0.00001	166814
8	0.0001	3	500	0.00001	1022020
10	0.00063	5	400	0.00029	1303226

TABLE IV
BEST-PERFORMING HYPERPARAMETERS FOR INN TRAINED ON A PLANAR ROBOT WITH REVOLUTE JOINTS

DoF	Learning rate	# Subnet layers	# Coupling layers	# Neurons per layer	Weight decay	# Trainable parameters
4	0.0009	5	9	100	0.00008	1108872
6	0.001	5	7	180	0.0003	2772084
8	0.0014	4	9	115	0.0004	997884
10	0.002	5	7	170	0.0005	2494380

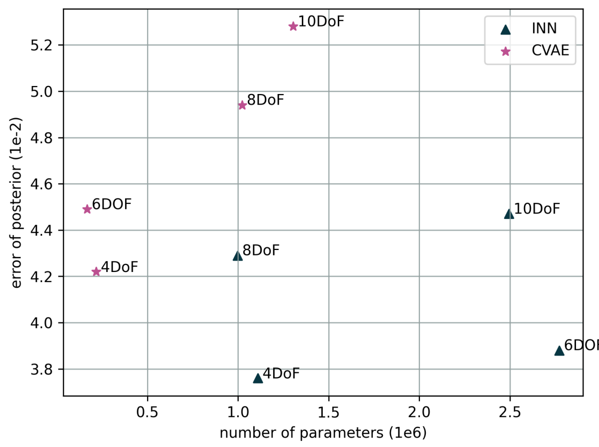


Fig. 2. Average posterior mismatch between the distribution generated by the model and the ground truth estimate obtained via rejection sampling.

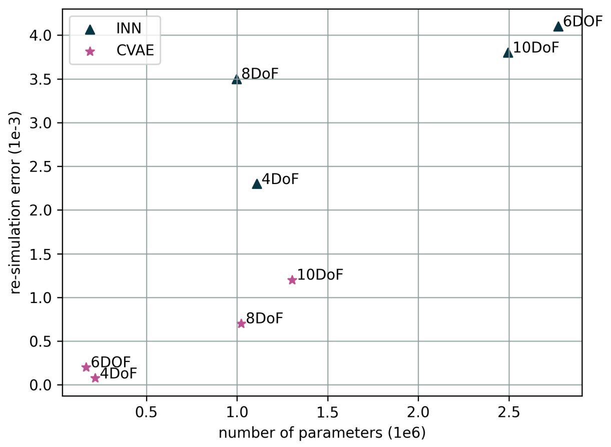
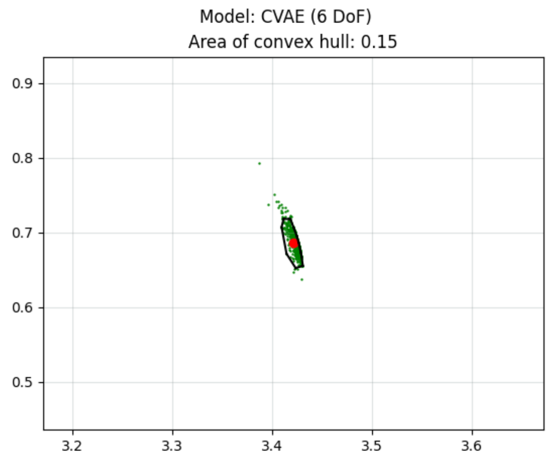


Fig. 3. Average re-simulation error as the mean squared distance between the ground truth end-effector position and the re-simulated end-effector position obtained from the predicted joint angles.

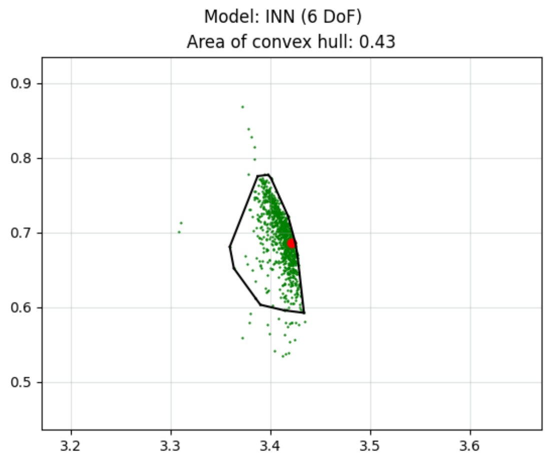
These results are illustrated in more detail with examples of 6DoF and 10DoF robots in Figures 4 and 5. A convex hull is drawn around the 97th percentile of the re-simulated points (green dots) and the corresponding ground truth end-effector position (red dot) for the cVAE and INN, respectively. As it can be seen, the area of this convex hull is larger for the INN (0.43 / 1.15) when compared with the cVAE (0.15 / 1.02). But despite of having a larger re-simulation error, the INN generates the full posterior distribution of the joint angles much better than the cVAE, as can be seen in Figures 6 and 7 for the 6DoF and 10 DoF robot, respectively.

CONCLUSION

In this work, we evaluated the feasibility of using neural networks to learn the posterior distribution of the hidden parameter space in the context of inverse kinematics problems

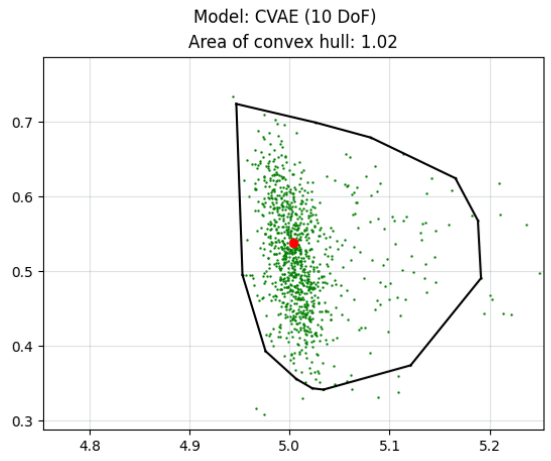


(a) cVAE

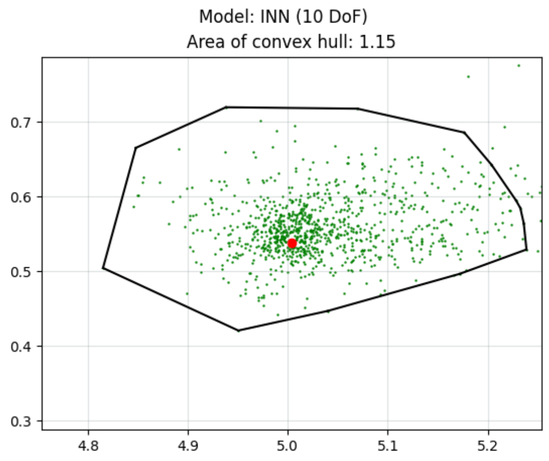


(b) INN

Fig. 4. Area of the convex hull of the 97th percentile of the re-simulated end-effector coordinates with the ground truth end-effector position at $(x, y) = [3.42, 0.68]$ and 1000 samples for a robot simulation with 6 DoF.



(a) cVAE



(b) INN

Fig. 5. Area of the convex hull of the 97th percentile of the re-simulated end-effector coordinates with the ground truth end-effector position at $(x, y) = [5.00, 0.53]$ and 1000 samples for a robot simulation with 10 DoF.

in robotics. For this, we developed simulations of planar robotic arm with an arbitrary number of revolute joints. Using the simulations, we studied Invertible Neural Networks [3] and compared them to conditional Variational Autoencoders [2]. We explored the performance of these architectures for both simple planar robot configurations with 4 revolute joints and more complex configurations with up to 10 revolute joints. To find the best set of hyperparameters we performed a hyperparameter search.

Using evaluation metrics defined in previous research [10], we found that cVAEs are better at predicting the values of the joint angles which result in a given the end-effector position. In contrast, our experiments show that INNs are able to better recover the full posterior distribution over the joint angle space.

Unlike stated in previous research [3], we had to perform an additional unsupervised backward training of the INN to match the performance of the cVAE. This resulted in considerably slower training times for the INN models. Moreover, the INN models tended to be more unstable during training. Despite these shortcomings, INNs are an interesting and powerful tool for recovering the full posterior parameters distribution and also for explicitly computing the posterior probability by using tractable Jacobians.

References

[1] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” 2014.
[2] K. Sohn, H. Lee, and X. Yan, “Learning structured output representation using deep conditional generative models,” in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee,

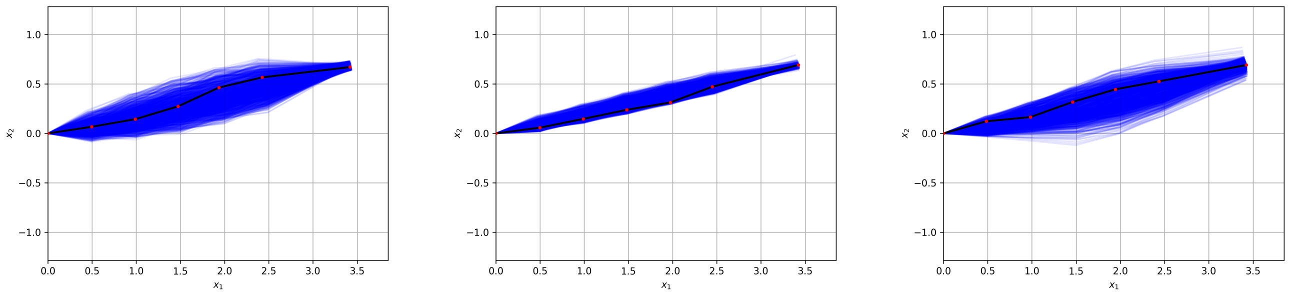

(a) Rejection Sampling (b) cVAE (c) INN

Fig. 6. Configurations of a planar manipulator with 6 revolute joints and end-effector position at $(x, y) = [3.42, 0.68]$. 1000 samples are drawn from each model's predicted posterior $\tilde{p}(x|y_{gt})$. One random sample configuration is highlighted. $e_{posterior} = 0.012$ for the cVAE and $e_{posterior} = 0.0068$ for the INN.

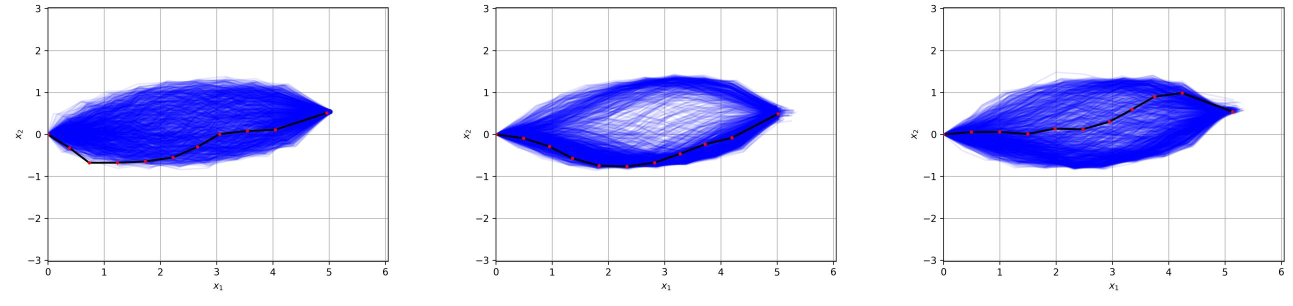

(a) Rejection Sampling (b) cVAE (c) INN

Fig. 7. Configurations of a planar manipulator with 10 revolute joints and end-effector position at $(x, y) = [5.00, 0.53]$. 1000 samples are drawn from each model's predicted posterior $\tilde{p}(x|y_{gt})$. One random sample configuration is highlighted. $e_{posterior} = 0.057$ for the cVAE and $e_{posterior} = 0.025$ for the INN.

M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015, pp. 3483–3491.

[3] L. Ardizzone, J. Kruse, S. J. Wirkert, D. Rahner, E. W. Pellegrini, R. S. Klessen, L. Maier-Hein, C. Rother, and U. Köthe, "Analyzing inverse problems with invertible neural networks," *CoRR*, vol. abs/1808.04730, 2018. [Online]. Available: http://arxiv.org/abs/1808.04730

[4] E. G. Tabak and E. Vanden-Eijnden, "Density estimation by dual ascent of the log-likelihood," *Communications in Mathematical Sciences*, vol. 8, no. 1, pp. 217 – 233, 2010. [Online]. Available: https://doi.org/

[5] E. Tabak and C. Turner, "A family of nonparametric density estimation algorithms," *Communications on Pure and Applied Mathematics*, vol. 66, no. 2, pp. 145–164, Feb. 2013, copyright: Copyright 2012 Elsevier B.V., All rights reserved.

[6] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," *arXiv preprint arXiv:1807.05118*, 2018.

[7] T. Head, MechCoder, G. Louppe, I. Shcherbatyi, fcharras, Z. Vinícius, cmmalone, C. Schröder, nel215, N. Campos, T. Young, S. Cereda, T. Fan, rene rex, K. K. Shi, J. Schwabedal, carlosdanielcsantos, Hvass-Labs, M. Pak, SoManyUsernamesTaken, F. Callaway, L. Estève, L. Besson, M. Cherti, K. Pfannschmidt, F. Linzberger, C. Cauet, A. Gut, A. Mueller, and A. Fabisch, "scikit-optimize/scikit-optimize: v0.5.2," Mar. 2018. [Online]. Available: https://doi.org/10.5281/zenodo.1207017

[8] graviraja, "pytorch-sample-codes," https://github.com/graviraja/pytorch-sample-codes, 2019.

[9] "Freia," https://github.com/VLL-HD/FrEIA/blob/master/FrEIA/, 2020.

[10] J. Kruse, L. Ardizzone, C. Rother, and U. Köthe, "Benchmarking invertible architectures on inverse problems," Tech. Rep. i, 2019.

[11] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *CoRR*, vol. abs/1411.1784, 2014. [Online]. Available: http://arxiv.org/abs/1411.1784

[12] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. J. Smola, "A kernel method for the two-sample problem," *CoRR*, vol. abs/0805.2368, 2008. [Online]. Available: http://arxiv.org/abs/0805.2368

[13] Y. Zhou, C. Barnes, L. Jingwan, Y. Jimei, and L. Hao, "On the continuity of rotation representations in neural networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[14] B. Choi and C. Lawrence, "Inverse kinematics problem in robotics using neural networks," Tech. Rep., 1992.

[15] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real NVP," *CoRR*, vol. abs/1605.08803, 2016. [Online]. Available: http://arxiv.org/abs/1605.08803