

REACHABILITY PROJECT LOG :)

CONTENTS

1. Problem Description	1
1.1. Problem inputs	1
1.2. Problem outputs	1
2. Simplified problem v2: Round thing with a rotary 2-link arm	2
2.1. Setup	2
2.2. Forward kinematics	2
2.3. Training	2
3. Simplified problem v1: Round thing with a fixed stick	2
3.1. Setup	2
3.2. Forward kinematics	2
3.3. Approaches	3
3.4. Evaluation	4
3.5. Extending to more general cases	4
3.6. Potential next-step extensions	4
4. Extra notes	4
4.1. Mode collapse	4
4.2. Conditional VAE theory	4
4.3. FrEIA INN theory	5
4.4. Normalizing flow	6
4.5. Conceptual comparison of different methods	6
4.6. Invertible neural networks vs. normalizing flow	6
References	7

1. PROBLEM DESCRIPTION

Our goal is to learn *where the robot should stand* given a desired grasp.

1.1. **Problem inputs.** Formulating the problem, we are given as inputs:

- $x_e \in SE(3)$: desired end-effector pose in world coordinates
- $b \in SE(2)$: wheeled mobile base pose (x, y, γ)
- $q \in \mathbb{R}^n$: arm joint configuration.

1.2. **Problem outputs.** We want to find a range of feasible whole-body configurations that satisfy the following requirements:

- (1) **IK feasibility:** forward kinematics(b, q) = x_e
- (2) **Visibility/sensing constraints:** The end-effector should lie within the camera's field of view and range (and un-occluded by the robot itself), i.e. $\text{visible}(b, q, x_e) = 1$
- (3) **Ball of feasible configurations:** We want a region in joint space around (b, q) that keeps the hand in approximately the right pose and satisfies the above constraints.

Our target is to model some distribution $p(b, q \mid x_e)$ subject to these constraints \uparrow , ideally with some notion of pose quality (visibility, manipulability, clearance, etc.).

Date: January 14, 2026.

2. SIMPLIFIED PROBLEM V2: ROUND THING WITH A ROTARY 2-LINK ARM

2.1. Setup.

2.2. Forward kinematics.

2.2.1. *fk_mse_from_qfeat loss function math.* Input to function: μ_q (μ_q = model’s prediction of the robot’s state) and H . We want to take this prediction, convert it into a valid physical configuration, calculate where the arms’ tip should be in the world and compare that to a target position H .

2.3. Training.

2.3.1. *Nearest neighbors baseline.* === Results: NNDeterministicLookup ===
hand_err/mean 0.014401
hand_err/median 0.013445
hand_err/p95 0.028278
coverage/max_gap_mean 6.283185
coverage/max_gap_p95 6.283185
coverage/kl_to_uniform 0.019053
stochasticity/var_Q_fixed_H 0.000000

2.3.2. Conditional VAE. Metrics to track:

- Total loss
- Reconstruction loss term
- KL loss term
- Mean of $|\mu|$: if near 0 always, encoder might be collapsing
- Mean of logvar: if very negative \rightarrow tiny standard deviation \rightarrow near-deterministic encoder
- Gradient norm: detect instability or dead training
- KL per dimension to show if any of the latent dimensions are “dead” (near-zero KL contribution)

2.3.3. Conditional INN.

3. SIMPLIFIED PROBLEM V1: ROUND THING WITH A FIXED STICK

3.1. **Setup.** Simple robot is a round thing with a fixed “stick” attached to it: a disk on the floor with a rigid stick of fixed length L glued to it, and the stick rotates with the disk.

- Configuration: $Q = x, y, \theta$ ($\mathbb{R}^2 \times S^1$)
- Hand target: $H = h_x, h_y$ (point in \mathbb{R}^2)

In this setup, $H \in \mathbb{R}^2$ is the desired 2D point on the floor where we want the tip of the stick to be.

3.2. Forward kinematics.

$$\text{hand}(x, y, \theta) := f(Q) = \begin{bmatrix} x \\ y \end{bmatrix} + L \begin{bmatrix} \cos \theta & \sin \theta \end{bmatrix}$$

So the IK constraint (“hand hits the target”) is:

$$H = \begin{bmatrix} h_x \\ h_y \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + L \begin{bmatrix} \cos \theta \\ \sin \theta \end{bmatrix}$$

Rearranging shows that as θ varies, the base center (x, y) traces out a circle of radius L around the target point H : the set of feasible base positions for the robot to reach H is a circle centered at H .

Ground truth feasible set for a fixed $H := \{Q : f(Q) = H\} = \{(h_x - L \cos \theta, h_y - L \sin \theta, \theta) : \theta \in [0, 2\pi)\}$

3.3. **Approaches.** We try the following approaches to model $p(Q | H)$:

(1) *Nearest neighbors* (baseline):

- Given a dataset $\mathcal{D} = \{(H_i, Q_i)\}_{i=1}^N$: for a query H' ,
 - (a) Find indices of the k nearest H_i to H' (under $\|H_i - H'\|$)
 - (b) Return their associated Q_i 's (either all or sample one etc.)
- For a new H' , we can return the Q_i associated to the nearest H_i , or sample from an empirical conditional distribution created based on the nearest neighbor graph: $\hat{p}_{\text{kNN}}(Q | H') = \sum_{j \in \mathcal{N}_k(H')} w_j(H') \delta(Q - Q_j)$ (where weight w_j is proportional to the distance between H' and H_i , i.e. $w_j \propto \exp(-\|H_j - H'\|^2 / \sigma^2)$).

(2) *Conditional VAE*: Introduce latent $z \in \mathbb{R}^d$: $z \sim \mathcal{N}(0, I)$, $Q \sim p_\theta(Q | H, z)$

- Train an encoder $q_\phi(z | Q, H)$ and decoder $p_\theta(Q | H, z)$ via ELBO loss:

$$\log p_\theta(Q | H) \geq \mathbb{E}_{z \sim q_\phi(z | Q, H)} [\log p_\theta(Q | H, z)] - \text{KL}[q_\phi(z | Q, H) || \mathcal{N}(0, I)]$$

- Potential decoder choice: output $\mu_\theta(H, z)$ and diagonal $\Sigma_\theta(H, z)$; use Gaussian likelihood:

$$\log p_\theta(Q | H, z) = \log \mathcal{N}(Q; \mu_\theta(H, z), \Sigma_\theta(H, z))$$

[TODO: Read into math more], notes in 4.2

- Advantages of cVAE for this problem:
 - Multi-modality via latent variable z (different z 's can map to different valid configurations on the circle) \rightarrow addresses mode collapse relative to deterministic nets [assuming we avoid posterior collapse (decoder ignores z)]
 - Smooth generalization in H : theoretically, the encoder/decoder can learn continuous maps in H -space, allowing better interpolation than the nearest neighbors approach.
- Note: θ is periodic, so training a Gaussian likelihood $p_\theta(Q | H, z)$ (in the decoder) on raw $\theta \in [0, 2\pi)$ is awkward because $\theta \approx 0$ and $\theta \approx 2\pi$ are close physically but far numerically ($\hat{\theta} = 2\pi - 0.01$ is a pretty good prediction for ground truth $\theta = 0.01$). To avoid this, instead of modeling $\theta \sim \mathcal{N}(\mu, \sigma^2)$, we can model a Gaussian over Cartesian coordinates in \mathbb{R}^2 :

$$(\cos \theta, \sin \theta) \sim \mathcal{N}(\mu_{cs}, \text{diag}(\sigma_{cs}^2))$$

We can recover θ via `atan2` at inference time.

(3) *Invertible NN* (FrEIA): [TODO: Read into math more], notes in 4.3

- **Representation:** To avoid periodicity issues in θ , we model:

$$Q_{\text{feat}} = (x, y, \cos \theta) \in \mathbb{R}^4, \quad H = (h_x, h_y) \in \mathbb{R}^2$$

- **Conditional invertible map:** We learn an invertible mapping (for each condition H)

$$f_\theta(\cdot; H) : \mathbb{R}^4 \leftrightarrow \mathbb{R}^4, \quad z = f_\theta(Q; H), \quad Q = f_\theta^{-1}(z; H)$$

with a simple base distribution $z \sim \mathcal{N}(0, I_4)$. In code this is implemented as a stack of conditional affine coupling blocks (FrEIA `SequenceINN + AllInOneBlock`) where each block receives H as a conditioning input.

- **Training:** Using the change-of-variables formula, the conditional density is:

$$p_\theta(Q | H) = p_Z(f_\theta(Q; H)) \left| \det \frac{\partial f_\theta(Q; H)}{\partial Q} \right|$$

Therefore:

$$\log p_\theta(Q | H) = \log p_Z(z) + \log |\det J_{f_\theta}(x; H)|$$

With $p_Z = \mathcal{N}(0, I)$, the per-example negative log-likelihood (dropping constants) is:

$$\mathcal{L}_{\text{NLL}}(Q, H) = \frac{1}{2} \|z\|^2 - \log |\det J_{f_\theta}(Q; H)|$$

We minimize $\mathbb{E}_{(Q, H) \sim \text{data}} |\mathcal{L}_{\text{NLL}}(Q, H)|$.

- **Sampling:** At test time, for a given H we sample:

$$z \sim \mathcal{N}(0, I_4), \quad Q = f_\theta^{-1}(z, H)$$

then convert $Q = (x, y, \cos \theta, \sin \theta)$ back to (x, y, θ) with $\theta = \text{atan2}(\sin \theta, \cos \theta)$.

- **Optional FK regularization (sample-space constraint):** Since the true conditional support lies on the IK manifold (all physically valid configurations for a given target must satisfy the IK constraint), we can add a penalty that directly enforces that generated samples reach H :

$$\mathcal{L} = \mathbb{E}_{(Q,H) \sim \mathcal{D}} [\mathcal{L}_{\text{NLL}}(Q, H)] + \lambda_{\text{FK}} \mathbb{E}_{H \sim \mathcal{D}, z \sim \mathcal{N}(0, I)} [\|f_{\text{FK}}(f_{\theta}^{-1}(z; H)) - H\|^2].$$

In our implementation we compute FK error on inverse samples $\hat{Q} = f_{\theta}^{-1}(z; H)$ during training (rather than on the training Q itself, as $\text{FK}(Q) = H$ by virtue of being in the dataset), so the regularizer shapes the *sampled* conditional distribution.

3.4. Evaluation. Assuming that the ground-truth $p^*(Q | H)$ is a uniform distribution over the feasible circle set, i.e.: $\theta \sim \text{Unif}[0, 2\pi) \rightarrow x = h_x - L \cos \theta, y = h_y - L \sin \theta$:

- Accuracy: $e_{\text{hand}}(Q, H) := \|f(Q) - H\|$
- Diversity/coverage: convert each sample to its implied angle on the circle:

$$\theta_{\text{implied}}^{(s)} := \text{atan2}(h_y - y^{(s)}, h_x - x^{(s)})$$

Check how well the empirical distribution over θ matches the target (here we assume $\theta \sim \text{Unif}[0, 2\pi)$): for histogram $\hat{p}(\theta)$, we can compute:

- KL divergence to uniform: $D_{\text{KL}}(\hat{p}(\theta) \parallel \text{Unif})$
- Max angle gap: sort angles $\theta^{(s)}$ around the circle, compute max gap $\Delta_{\text{max}} = \max_i (\theta_{i+1} - \theta_i)$
Large Δ_{max} = model is missing big arcs (collapse)
- “Uses latent” test (for cVAE/cINN): fix H , sample many z ’s, then measure output variance $\text{Var}(Q | H)$. If the model ignores z , this variance will be small.
[\[↓ TODO: Implement\]](#)
- Generalization in H : make a test set where H is (1) in-distribution, (2) near boundary, (3) out-of-distribution (slightly outside training workspace). Compare success and coverage!
- Inference speed per sample

3.5. Extending to more general cases. Ideally, would be nice if this toy problem gave us an idea of the advantages/disadvantages of different approaches in regards to the following challenges:

- *Mode collapse*: Test: fixed H , sample 1k solutions, compute Δ_{max} , KL-to-uniform, etc.
- *Generalization*: Test: distribution of $e_{\text{hand}} = \|f(Q) - H\|$
- *Bias from data collection* (forward vs. inverse sampling):

3.6. Potential next-step extensions.

- Disconnected feasible sets: modify toy with a “visibility” constraint such as

$$\text{visible}(Q, H) = 1 \iff \theta \in [\alpha_1, \beta_1] \cup [\alpha_2, \beta_2]$$

Now the true $p^*(Q | H)$ is two separated modes. **Test:** cluster sampled angles into arcs and compute mode recall: fraction of samples that land in each arc.

- Scalability with dimension: add DoF to the toy (e.g., 2-link arm, variable stick length). **Test:** track how coverage/error degrades with dimension.

4. EXTRA NOTES

4.1. Mode collapse. A generative model has mode collapse if, for a fixed condition H , the samples $Q \sim \hat{p}(Q | H)$ cover only a small subset of the true support of $p^*(Q | H)$.

- In this toy problem, the support is the full circle parameterized by θ . Mode collapse looks like the model always outputting $\theta \approx 0$ (i.e. stands in one favorite location) or outputting only a few discrete angles, ignoring the rest the circle, etc.

4.2. Conditional VAE theory. The goal is to model a multi-modal conditional distribution over robot configurations Q given target H : $p^*(Q | H)$. In our simple robot setup, the true conditional has a 1D manifold of solutions (a circle in (x, y) paired with corresponding θ), so it’s not unimodal in the usual sense (rather than being a single blob in \mathbb{R}^d , for fixed H , probability mass $p^*(Q | H)$ is concentrated along a ring in (x, y) with a corresponding orientation θ at each point).

4.2.1. *Motivation.* Because $p^*(Q | H)$ is not unimodal, if we model $p(Q | H)$ as a single Gaussian/predict a single mean, we face mean collapse: the mean of points around a circle is the center of the circle, but the center is invalid. So the “best” unimodal prediction is often physically invalid/low-probability under the true distribution.

4.2.2. *Model architecture.* We introduce latent $z \in \mathbb{R}^{d_z}$, which ideally will be the variable that “chooses” which solution the model means among the many possible solutions on the ring for a given H . Define a prior $p(z) = \mathcal{N}(0, I)$. There are two parts to the model (decoder and encoder):

- Decoder (generative model): The decoder is a conditional distribution $p_\theta(Q | H, z)$. For the architecture of the decoder, we use the diagonal Gaussian ([potential next step – try other architectures](#)):

$$p_\theta(Q | H, z) = \mathcal{N}(Q; \mu_\theta(H, z), \text{diag}(\sigma_\theta^2(H, z)))$$

where $\mu_\theta(H, z)$ and σ_θ^2 are the predicted mean/variance of each coordinate of Q .

- Interlude: $p_\theta(Q, z | H) = p(z | H)p(Q | z, H) = p(z)p_\theta(Q | H, z)$ [assuming $p(z | H) = p(z)$]. Bayes’ rule gives:

$$p_\theta(z | Q, H) = \frac{p_\theta(Q, z | H)}{p_\theta(Q | H)} = \frac{p(z)p_\theta(Q | H, z)}{p_\theta(Q | H)}$$

Expanding the denominator: $p_\theta(Q | H) = \int p(z')p_\theta(Q | H, z') dz'$ (consider all possible latent explanations z' , weight how likely each would produce Q , then sum them up). Plugging this in,

$$p_\theta(z | Q, H) = \frac{p(z)p_\theta(Q | H, z)}{\int p(z')p_\theta(Q | H, z') dz'}$$

The denominator $\int p(z')p_\theta(Q | H, z') dz'$ is intractable: $p_\theta(Q | H, z') = \mathcal{N}(Q; \mu_\theta(H, z'), \Sigma_\theta(H, z'))$, so we effectively have

$$p(z')p_\theta(Q | H, z') \Rightarrow \mathcal{N}(z'; 0, 1) \times \mathcal{N}(Q; \mu_\theta(H, z'), \Sigma_\theta(H, z'))$$

For fixed Q and H , $\mathcal{N}(Q; \mu_\theta(H, z'), \Sigma_\theta(H, z'))$ becomes some complicated nonnegative function of $z' \rightarrow g(z')$. Then,

$$p_\theta(Q | H) = \int \mathcal{N}(z'; 0, 1)g(z') dz'$$

This is very hard to compute analytically: recall the multivariate Gaussian density is given by:

$$f_{\mu, \Sigma}(Q) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp(-\frac{1}{2}(Q - \mu)^\top \Sigma^{-1}(Q - \mu))$$

Here, we plug in $\mu = \mu_\theta(H, z'), \Sigma = \Sigma_\theta(H, z')$. These are neural-net outputs, i.e. complicated nonlinear functions of z' , which makes the integral difficult to compute.

Since the true posterior is intractable, introduce a tractable approximation $q_\phi(z | Q, H) : (Q, H) \mapsto p(z)$ which essentially answers “If this Q came from some latent z , what must that z have been?”

- Encoder (inference model): The encoder is the conditional distribution $q_\phi(z | Q, H)$.

4.3. **FrEIA INN theory.** The goal is to learn $p(Q | H)$, where $Q = (x, y, \cos \theta, \sin \theta) \in \mathbb{R}^4$ (equivalent form of (x, y, θ)) and $H = (h_x, h_y) \in \mathbb{R}^2$. A normalizing flow models an invertible map (for each fixed condition h):

$$f_\theta(\cdot; h) : \mathbb{R}^4 \rightarrow \mathbb{R}^4, \quad z = f_\theta(q; h), \quad q = f_\theta^{-1}(z; h)$$

We pick the base density for z to be the multivariate standard normal:

$$p_Z(z) = \mathcal{N}(0, I_4)$$

Then, $p_\theta(q | h)$ can be defined via “push q through f to get z , and measure how likely that z is, corrected by volume change” \rightarrow correction is the Jacobian determinant.

4.3.1. *Hand-wavy intuition.* For each condition h , the true data q lives in some weird, potentially multi-modal shape (ring of solutions in our simplified problem). A Gaussian cannot represent that shape directly.

- *Core idea:* Instead of trying to fit a complicated distribution directly, we learn a warp of space that makes the complicated shape look Gaussian.
 - Forward direction: $z = f_\theta(q; h)$ takes real samples q and maps them into “Gaussian space”
 - Training enforces: “after mapping, these z ’s should look like samples from $\mathcal{N}(0, I)$ ”
 - Warping space changes volume \rightarrow probability densities change:
 - * If f expands a region of q -space, then points there become less dense in q -space.
 - * If f compresses a region, points become more dense.

After learning an invertible NN model, we can generate realistic q by:

- (1) Sample $z \sim \mathcal{N}(0, I)$
- (2) Invert: $q = f_\theta^{-1}(z; h)$

Fun way of thinking about it: Imagine data for q ’s is shaped like a bent pretzel in \mathbb{R}^D . A flow learns a continuous, invertible deformation of space that morphs the pretzel into a D -dim. Gaussian blob. Forward pass = “unbend the pretzel into a Gaussian.” Inverse pass = “bend a Gaussian blob back into a pretzel.” The Jacobian term is bookkeeping of how the deformation stretches space.

4.3.2. *Why layers need to be invertible.*

- (1) Invertible layers allow flow models to compute an exact probability for any observed training point: if Q is a real solution from the dataset for some H , we can compute exactly how likely the model thinks it is (flows are the only model out of GANs, VAEs, and diffusion models that can compute the exact log-likelihood of a new sample)

$$\Pr(\mathbf{f}(\mathbf{Q}) \mid \mathbf{H}, \phi) = \left| \frac{\partial \mathbf{f}(\mathbf{z}, \mathbf{H}, \phi)}{\partial \mathbf{z}} \right|^{-1} \cdot \Pr(\mathbf{z}) \text{ where } \mathbf{z} = \mathbf{f}^{-1}(\mathbf{x}, \mathbf{H}, \phi) \text{ and } \phi = \text{learned model params}$$

- (2) Invertibility allows sampling to be one inverse pass rather than iterative denoising, sampling chains, etc.: once trained, we can directly (1) sample $z \sim \mathcal{N}(0, I)$ (2) compute $Q = f_\theta^{-1}(z; H)$

4.3.3. *More concrete math.* In separate PDF (flow model FrEIA implementation notes.pdf).

4.4. **Normalizing flow.** In separate PDF (normalizing flow notes.pdf).

4.5. **Conceptual comparison of different methods.** Given the data is shaped like a pretzel in $\mathbb{R}^D \dots$

4.5.1. *Flow.* Learn an invertible deformation conditioned on H that straightens the pretzel into a Gaussian $\mathcal{N}(0, I_D)$. Sampling works by running the inverse deformation on $z \sim \mathcal{N}(0, I_D)$ and H to obtain Q^* .

4.5.2. *cVAE.* Learn a conditional generator that uses Gaussian knobs to choose how to output the pretzel; training teaches the knobs to stay Gaussian while reconstructing data.

4.5.3. *Diffusion.* Instead of unbending the pretzel into a Gaussian in one shot, diffusion dissolves the pretzel into pure noise by gradually adding noise, then learns to sculpt noise back into the pretzel step-by-step.

4.6. **Invertible neural networks vs. normalizing flow.** Invertible neural networks are a class of deep networks that approximate bijective functions and are characterized by a forward mapping that can be inverted. All normalizing flow networks are invertible neural networks.

4.6.1. *INNs as generative models.* Because INNs learn a transformation between the input data distribution and a simple prior distribution (typically a normal distribution), the inverse mapping automatically acts as a generator, converting the simple distribution into samples resembling the input data.

4.6.2. *INNs that are not normalizing flow models.* Invertible neural networks that are not flow models exist (TODO: Read into this paper [1]).

REFERENCES

- [1] Lynton Ardizzone, Jakob Kruse, Sebastian J. Wirkert, Daniel Rahner, Eric W. Pellegrini, Ralf S. Klessen, Lena Maier-Hein, Carsten Rother, and Ullrich Köthe. Analyzing inverse problems with invertible neural networks. *CoRR*, abs/1808.04730, 2018.