PROBABILISTIC TRAJECTORY RECONSTRUCTION FROM KEYFRAMES

CHECKPOINT 2 FOR CS 145 BDL FINAL PROJECT

Hyunsu Lee **Larry Qiu** **Xianmai Liang**

Sharing permission statement:
Course staff can share this report to future students of similar classes

1 INTRODUCTION

In robotics applications where dexterous, precise, or responsive motion is desired, high frequency control is often required. These applications include robotic arms used in manipulation tasks, legged robots for locomotion across diverse terrains, or high throughput industrial robots. However, when control becomes high frequency, the entire pipeline needs to be scaled to match: sensor polling rates increase, datasets become bloated, models need to be scaled up, and control hardware must be more powerful. One method of avoiding these drawbacks is to approximate high frequency motion with keyframes, which are snapshots of the robot's position at sparse timestamps (Shi et al., 2023; Belkhale et al., 2023). This enables data collection, storage, modeling, and prediction to operate on much sparser data, which can be advantageous in limited compute, bandwidth, or storage environments.

However, a key step in this method is reconstructing the high frequency samples from the keyframes for control. In our project, we explore a probabilistic approach towards this problem, where we model a trajectory distribution given a set of keyframes. A probabilistic approach has several benefits. For example, the distribution can be measured to determine the model's degree of confidence and whether or not more waypoints are necessary. The distribution can also be used to sample trajectories biased towards immediate needs like safety requirements or energy efficiency.

In our project, we experiment with two methods. As a baseline, we consider a per-joint Gaussian process (GP) model that interpolates between keyframes, where the dataset is used to optimize parameters. For our upgrade, we use a masked variational autoencoder (VAE), where the keyframes are represented as an entire trajectory with the non-keyframe timestamps masked out. This technique has been applied in other domains (Ma et al., 2019; Nazabal et al., 2020; Collier et al., 2020), but we extend it to the robotics task. We hypothesize that compared to the per-joint GP, the masked VAE has a lower MSE on our block-picking dataset at low keyframe rates ($\leq$ 1 Hz), because the VAE can model correlations and patterns in the data that the gaussian process cannot represent. At high rates we expect the GP to remain competitive, since dense observations make the learned prior less important. We're also interested in comparing the methods' coverage, which measures how well the predicted distribution captures the true dataset distribution, but we're not sure how either method will compare.

2 DATA AND ANALYSIS PLAN

2.1 DATASET

We use a block-picking dataset collected on the SO-101 low-cost robotic arm, which was created by a team member for a previous project. The dataset consists of 166 episodes of the robot arm picking up a wooden block, recorded at 20 Hz from the follower arm during teleoperation. We use the five arm joints (shoulder pan, lift, elbow flex, wrist flex, wrist roll), discarding the camera feed also present in the dataset. We denote the raw trajectory of episode n as $x^{(n)} \in \mathbb{R}^{T_n \times 5}$ with $T_n \in [46, 197]$ (median 112 timesteps, $\sim$5.6 s), where each entry is a real-valued angle in degrees. Figure 1 shows end-effector trajectories for 20 sample episodes.

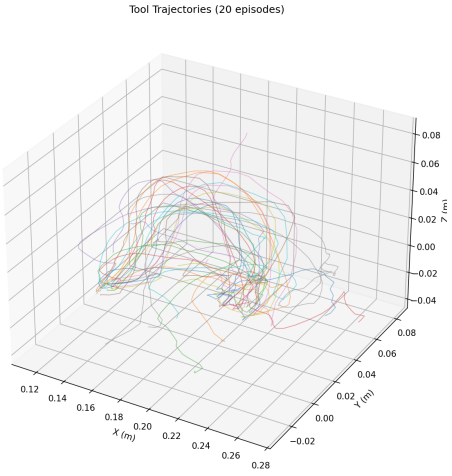

Figure 1: End-effector trajectories of 20 sample episodes, computed from the joint-angle sequences via forward kinematics.

We center each joint using the per-joint dataset mean $\mu \in \mathbb{R}^5$: $\tilde{x}_t^{(n)} = x_t^{(n)} - \mu$. We don't rescale by standard deviation, because all five dimensions share the same physical unit (degrees) and a per-joint rescaling would discard relative amplitude information. We use integer timestamps $t \in \{0, \ldots, T_n - 1\}$ since the sample rate is constant (each increment corresponds to 0.05 seconds).

2.2 Experimental Plan

Inputs and outputs. The task input for an episode n is a list of keyframe timesteps $\mathcal{O}^{(n)} = [t_0, t_1, \ldots, t_{|\mathcal{O}^{(n)}|-1}] \subseteq \{0, \ldots, T_n - 1\}$ together with the joint angle vectors at those timesteps, $\tilde{x}^{(n)}_{\mathcal{O}^{(n)}} = [\tilde{x}^{(n)}_{t_0}, \tilde{x}^{(n)}_{t_1}, \ldots] \in \mathbb{R}^{|\mathcal{O}^{(n)}| \times 5}$. The task output is a probability distribution $p_\theta(\tilde{x}^{*(n)} \mid \mathcal{O}^{(n)}, \tilde{x}^{(n)}_{\mathcal{O}^{(n)}})$ over a predicted full trajectory $\tilde{x}^{*(n)} \in \mathbb{R}^{T_n \times 5}$. The star superscript denotes the prediction of the model, in comparison to the ground truth $\tilde{x}^{(n)}$.

Keyframe mask construction. To create the keyframe timestamps $\mathcal{O}^{(n)}$, we uniformly subsample the input trajectory. Given a rate f_{freq} in Hz, let $s = \lfloor 20/f_{\text{freq}} \rfloor$ and $\mathcal{O}^{(n)} = \{0, s, 2s, \ldots\} \cup \{T_n - 1\}$. For this project, we use $f_{\text{freq}} \in \{0, 0.2, 0.5, 1, 2, 5\}$ Hz for evaluation, where $f_{\text{freq}} = 0$ corresponds to the start only case, $\mathcal{O}^{(n)} = \{0\}$.

Train / validation / test split. We split the 166 episodes at the episode level into 60% train (99), 20% validation (33), and 20% test (34). The baseline uses the training split only for hyperparameter selection; the upgrade uses it for model fitting. The numbers below used to select hyperparameters are all validation metrics.

Metrics. For this project, we consider the following metrics:

- **Mean Squared Error (MSE).** The mean-squared error between the ground truth and the mean of the model distribution, estimated with Monte Carlo:

$$\text{MSE}^{(n)} = \frac{1}{T_n} \sum_{t=0}^{T_n - 1} \mathbb{E}_{p_\theta}\left[\left\|\tilde{x}_t^{*(n)} - \tilde{x}_t^{(n)}\right\|_2^2\right].$$

- **Coverage**, for $\alpha \in \{50\%, 80\%, 95\%\}$. The fraction of ground-truth datapoints that fall in the α-level credible interval, also estimated with Monte Carlo:

$$\text{Cov}_\alpha^{(n)} = \frac{1}{5T_n} \sum_{t,j} \mathbf{1}\left[\tilde{x}_{t,j}^{(n)} \in \left[\text{quantile}_{(1-\alpha)/2},\ \text{quantile}_{(1+\alpha)/2}\right]\right],$$

  where quantile_p denotes the p-th quantile of the distribution of $\tilde{x}_{t,j}^{*(n)}$; a model that accurately models the distribution has $\mathbb{E}[\text{Cov}_\alpha^{(n)}] = \alpha$.

We chose these metrics because they can be evaluated across both models using monto carlo methods (whereas posterior log likelihood or ELBO cannot be). Additionally, using both MSE and coverage allows us to evaluate both the accuracy of the prediction and the accuracy of the uncertainty estimation, which is desired for our task.

3 BASELINE: INDEPENDENT PER-JOINT GAUSSIAN PROCESS

3.1 PROBABILISTIC MODEL

The task uses discrete timesteps, but a GP models a distribution over continuous functions. We therefore use a proxy function f that maps a real-valued timestep to each joint angle (f is used only in this section). For each joint $j \in \{1, \dots, 5\}$ we define $f_j^{(n)} : \mathbb{R} \to \mathbb{R}$ with $\tilde{x}_{t,j}^{*(n)} = f_j^{(n)}(t)$.

We use a zero-mean GP prior on $f_j^{(n)}$ with the kernel k_j:

$$f_j^{(n)} \;\sim\; \mathcal{GP}\big(0,\; k_j(t, t')\big).$$

The five joints are individually modeled with independent GPs. Then, we define the likelihood based on the keyframes using a gaussian distribution with a fixed standard deviation τ:

$$p\Big(\tilde{x}_{t_i,j}^{(n)} \,\Big|\, f_j^{(n)}(t_i)\Big) \;=\; \mathcal{N}\Big(\tilde{x}_{t_i,j}^{(n)} \,\Big|\, f_j^{(n)}(t_i),\; \tau^2\Big), \quad t_i \in \mathcal{O}^{(n)}.$$

We test four kernels, each with length scales $L \in \{3, 10, 30, 100, 300\}$ timesteps.

- Squared exponential (RBF), producing infinitely differentiable samples:

$$k(t, t') = \sigma_f^2 \exp\left(-\frac{(t - t')^2}{2L^2}\right).$$

- Matérn $\nu = 0.5$, producing non-differentiable samples:

$$k(t, t') = \sigma_f^2 \exp\left(-\frac{|t - t'|}{L}\right).$$

- Matérn $\nu = 1.5$, producing once-differentiable samples:

$$k(t, t') = \sigma_f^2 \left(1 + \frac{\sqrt{3}\,|t - t'|}{L}\right) \exp\left(-\frac{\sqrt{3}\,|t - t'|}{L}\right).$$

- Matérn $\nu = 2.5$, producing twice-differentiable samples (between ν=1.5 and RBF in smoothness):

$$k(t, t') = \sigma_f^2 \left(1 + \frac{\sqrt{5}\,|t - t'|}{L} + \frac{5\,(t - t')^2}{3L^2}\right) \exp\left(-\frac{\sqrt{5}\,|t - t'|}{L}\right).$$

L controls how quickly correlation decays with timestep separation while σ_f^2 scales the prior's variance to match each joint's amplitude.

3.2 ESTIMATION OBJECTIVE AND ALGORITHM

Using Bayes' rule, the posterior over $f_j^{(n)}$ given the observed keyframes is also a GP with closed-form mean and variance at any query timestep t_*. Let $\tilde{x}_{\mathcal{O}^{(n)},j}^{(n)} \in \mathbb{R}^{|\mathcal{O}^{(n)}|}$ denote the vector of observations of joint j at the keyframe timesteps. Then

$$p\big(f_j^{(n)}(t_*) \,\big|\, \tilde{x}_{\mathcal{O}^{(n)},j}^{(n)}\big) \;=\; \mathcal{N}\big(f_j^{(n)}(t_*) \,\big|\, \mu_*,\, \sigma_*^2\big),$$

$$\mu_* \;=\; K_{*N}\big(K_{NN} + \tau^2 I\big)^{-1}\, \tilde{x}_{\mathcal{O}^{(n)},j}^{(n)}, \qquad \sigma_*^2 \;=\; k_j(t_*, t_*) \;-\; K_{*N}\big(K_{NN} + \tau^2 I\big)^{-1} K_{N*}.$$

Concretely, $K_{NN} \in \mathbb{R}^{|\mathcal{O}^{(n)}| \times |\mathcal{O}^{(n)}|}$ has entries $[K_{NN}]_{ab} = k_j(t_a, t_b)$ for $t_a, t_b \in \mathcal{O}^{(n)}$; $K_{*N} \in \mathbb{R}^{1 \times |\mathcal{O}^{(n)}|}$ has entries $k_j(t_*, t_a)$; and $K_{N*} = K_{*N}^\top$. The full predicted trajectory is created by evaluating μ_*, σ_*^2 at each query timestep $t_* = 0, \dots, T_n - 1$.

3.3 Implementation

The model was implemented in Python with NumPy and SciPy (where `scipy.special.kv` was used for the Bessel function in the Matérn kernels). This is based off of the code from Homework 1. Plotting was done using Matplotlib. No GP library (GPy, GPyTorch, sklearn) is used. Experiments were done on a laptop CPU, with each configuration experiment taking a few seconds.

3.4 Hyperparameter plan

We plan on using a grid search over the different kernel types and length scales, with the noise standard deviation τ and kernel variance $\sigma_{f,j}^2$ fixed based on the training data. The kernel type and length scale are the most important hyperparameters to tune, since they need to be matched to our dataset's smoothness and temporal correlation.

Hyperparameter	Value(s)	How chosen
τ (noise std)	2.0°	Fixed, estimated from the training data.
$\sigma_{f,j}^2$ (per joint)	Fixed to match per-joint standard deviation.	
Kernel	{SqExp, Mat. 0.5, 1.5, 2.5}	Grid search to minimize median validation MSE.
Length scale L	$\{3, 10, 30, 100, 300\}$ timesteps	Grid search to minimize median validation MSE.

4 Upgrade: Masked Variational Autoencoder

4.1 Probabilistic model

In a variational autoencoder, the goal is to map a latent distibution into the dataset distribution. Here, each latent code $z \in \mathbb{R}^{d_z}$ maps to an entire trajectory, with a Gaussian prior $z \sim \mathcal{N}(0, I_{d_z})$. The decoder then maps each latent code to a gaussian distribution with learned mean and variance at each timestep:

$$p_\theta\big(\tilde{x}^{*(n)} \,\big|\, z\big) \;=\; \prod_{t=0}^{T_n-1} \mathcal{N}\big(\tilde{x}_t^{*(n)} \,\big|\, \mu_\theta(z,t),\; \mathrm{diag}(\sigma_\theta^2(z,t))\big).$$

Note that as the entire trajectory is modeled as a single output tensor, all samples need to have an equal length, so we pad or truncate trajectories to a fixed length $T = 200$ timesteps (10 seconds).

Based on Ma et al. (2019); Nazabal et al. (2020); Collier et al. (2020), we give the encoder both a copy of the masked keyframe values (where non-keyframe timesteps are zeroed out) and a binary mask indicating which timesteps are keyframes, allowing it to distinguish between zero values that are true keyframes and zero values that are masked-out non-keyframes.

$$\hat{x}_t^{(n)} = \begin{cases} \tilde{x}_t^{(n)}, & t \in \mathcal{O}^{(n)} \\ \mathbf{0}, & \text{otherwise} \end{cases} \in \mathbb{R}^{T\times 5}, \qquad m_t^{(n)} = \mathbf{1}\big[t \in \mathcal{O}^{(n)}\big] \in \{0,1\}^T,$$

and define

$$q_\phi\big(z \,\big|\, \mathcal{O}^{(n)},\, \tilde{x}_{\mathcal{O}^{(n)}}^{(n)}\big) \;=\; \mathcal{N}\Big(z \,\Big|\, \mu_\phi(\hat{x}^{(n)}, m^{(n)}),\; \mathrm{diag}(\sigma_\phi^2(\hat{x}^{(n)}, m^{(n)}))\Big).$$

4.2 Estimation objective and algorithm

We maximize the reconstruction ELBO, which was based on the β-VAE method proposed by (Higgins et al., 2017):

$$\mathcal{L}(\theta,\phi;\, \tilde{x}^{(n)}, \mathcal{O}^{(n)}) = \mathbb{E}_{z\sim q_\phi(z\mid \mathcal{O}^{(n)},\, \tilde{x}^{(n)}_{\mathcal{O}^{(n)}})}\left[\, \frac{1}{|\mathcal{O}^{(n)}|} \sum_{t\in\mathcal{O}^{(n)}} \log p_\theta\big(\tilde{x}_t^{(n)} \,\big|\, z\big) \,\right] - \beta\, \mathrm{KL}\Big(q_\phi(z \mid \mathcal{O}^{(n)},\, \tilde{x}^{(n)}_{\mathcal{O}^{(n)}}) \,\Big\|\, \mathcal{N}(0, I_{d_z})\Big),$$

where β is the β-VAE weight.

We optimized the model using stochastic gradient ascent with Adam (Kingma & Ba, 2014) with the reparameterization trick to differenciate through Normal sampling.

4.3 Implementation

Both networks are MLPs implemented in PyTorch without any VAE-specific library. Let $[\hat{x}^{(n)} \,\|\, m^{(n)}] \in \mathbb{R}^{T\times 6}$ denote the per-timestep concatenation of the feature and mask, and let $\mathrm{PE}(t) \in \mathbb{R}^{16}$ be a sinusoidal positional encoding. The specific encoder and decoder structures are as follows:

Encoder.

$$
\begin{aligned}
h_0 &= \mathrm{flatten}\big([\hat{x}^{(n)} \,\|\, m^{(n)}]\big) \in \mathbb{R}^{6T},\\
h_1 &= \mathrm{ReLU}(W_1 h_0 + b_1), \qquad W_1 \in \mathbb{R}^{256\times 6T},\\
h_2 &= \mathrm{ReLU}(W_2 h_1 + b_2), \qquad W_2 \in \mathbb{R}^{256\times 256},\\
\mu_\phi &= W_3^{\mu} h_2 + b_3^{\mu}, \qquad W_3^{\mu} \in \mathbb{R}^{d_z\times 256},\\
\log \sigma_\phi &= W_3^{\sigma} h_2 + b_3^{\sigma}, \qquad W_3^{\sigma} \in \mathbb{R}^{d_z\times 256}.
\end{aligned}
$$

Decoder.

$$
\begin{aligned}
g_0(t) &= [\, z \,\|\, \mathrm{PE}(t) \,] \in \mathbb{R}^{d_z+16},\\
g_1(t) &= \mathrm{ReLU}(V_1 g_0(t) + c_1), \qquad V_1 \in \mathbb{R}^{256\times(d_z+16)},\\
g_2(t) &= \mathrm{ReLU}(V_2 g_1(t) + c_2), \qquad V_2 \in \mathbb{R}^{256\times 256},\\
\mu_\theta(z,t) &= V_3^{\mu} g_2(t) + c_3^{\mu}, \qquad V_3^{\mu} \in \mathbb{R}^{5\times 256},\\
\log \sigma_\theta(z,t) &= V_3^{\sigma} g_2(t) + c_3^{\sigma}, \qquad V_3^{\sigma} \in \mathbb{R}^{5\times 256}.
\end{aligned}
$$

In our preliminary tests, we found that training runs on a single laptop CPU and takes about 10 minutes for 300 epochs. We expect to be able to run the hyperparameter sweep within a few hours on the CPU, but we can also use a GPU if necessary.

4.4 Hyperparameter plan

There are many hyperparameters to tune, including the latent dimension d_z, hidden architecture, β-VAE weight, and learning rate. We're not which values will impact the results the most, so we plan on doing a grid search over all of them, optimizing for the median validation ELBO across episodes. The current values are based on selecting reasonable values from the papers and adjusting based on preliminary tests. However, even without precise tuning the upgrade already outperforms the baseline, as shown in the results section, so we don't expect the final results to be too sensitive to hyperparameter selection.

Hyperparameter	Candidate values	How chosen
Latent dim d_z	$\{4, 8, 16, 32\}$	Grid search, optimizing validation ELBO.
Hidden architecture	$\{[128, 128], [256, 256], [256, 256, 256]\}$	Grid search, optimizing validation ELBO.
β (KL weight)	$\{0.1, 1.0, 5.0\}$	Grid search, optimizing validation ELBO.
Learning rate	$\{10^{-4}, 10^{-3}\}$	Grid search, optimizing validation ELBO.
Batch size	32	Fixed.
Epochs	300	Fixed.
Train mask rate f_{freq}	$\mathrm{Unif}\{0, 0.2, 0.5, 1, 2, 5\}$ Hz	Fixed to match evaluation.

5 Results and Analysis (Draft)

Figure 2 plots the median validation MSE of the baseline GP and the VAE upgrade against the keyframe rate f_{freq} on the 33-episode validation set. For each rate we created the keyframe mask

as described in Section 2.2, ran both methods on the same mask, and computed MSE for every validation episode. The colored region show the 25th–75th percentile of MSE across episodes. This plot directly addresses the research hypothesis, which predicts that the VAE should have lower MSE than the GP at rates ≤ 1 Hz (where the GP's zero-mean prior has nothing local to interpolate between) and that the GP should stay competitive at rates ≥ 2 Hz (where the kernel's length scale covers the inter-keyframe gap). The two curves cross near 1 keyframe/s in the expected direction: the VAE wins at sparse rates and the GP wins at dense rates.

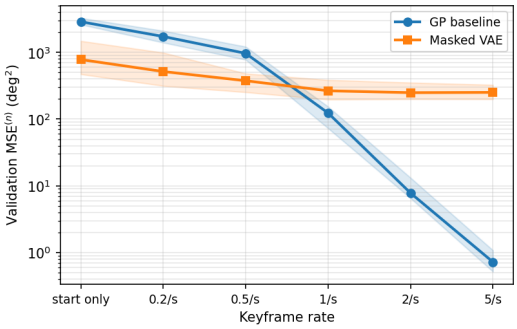

Figure 2: Median validation MSE (log scale) vs. keyframe rate across 33 episodes for the tuned GP (Matérn ν=2.5, L=10) and the masked VAE (d_z=16, [256, 256], β=1.0, 300 epochs). Bands: 25th–75th percentile across episodes.

A Changelog

- We reframed the project based on feedback from checkpoint 1, focusing on the keyframe reconstruction task to better motivate the problem and approach.
- We also made the notation more consistant and highlighted the distinction between the discrete task definition and the continuous function modeling of the GP.
- We removed the energy-based evaluation metric for simplicity for now.

B Future Plan

Roadblocks.

- Based on preliminary tests, we found that the VAE's coverage was much worse that the GP baseline, indicating that the VAE is overconfident in its predictions and that the posterior is collapsing. We're not sure why this is occuring, but we will try adjusting the model size and seeing if it is overfitting or if the issue is more fundamental to the masked VAE approach.

Milestones.

- Apr 20 — VAE hyperparameter sweep (d_z, hidden, β, learning rate). (Larry)
- Apr 25 — Discuss results and write sections in final report (All)
- Apr 25 — Test model on robot: replay a full teleoperated episode as well as a inferred trajectory based on keyframes and compare (Larry)
- May 1 — Possibly re-integrate the energy consideration from previous checkpoint as another method of comparison. (Chris)
- May 3 — Final figures, poslish paper. (Hyun)
- May 7 — Submit Final Report. (All)

C Questions

- Is there any obvious reason why the VAE's coverage would be worse than the GP's? We expected the VAE to be more flexible and thus better calibrated (especially since the vari-

ance is also learned), but it seems to be overconfident. Is this a common issue with VAEs or is it something specific to our implementation?

REFERENCES

C. Ma et al. *EDDI: Efficient Dynamic Discovery of High-Value Information with Partial VAE*. ICML 2019. https://arxiv.org/abs/1809.11142

A. Nazabal, P. M. Olmos, Z. Ghahramani, I. Valera. *Handling Incomplete Heterogeneous Data using VAEs*. Pattern Recognition, 2020. https://arxiv.org/abs/1807.03653

M. Collier, A. Nazabal, C. K. I. Williams. *VAEs in the Presence of Missing Data*. ICML Artemiss Workshop, 2020. https://arxiv.org/abs/2006.05301

I. Higgins et al. *β-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework*. ICLR 2017.

L. X. Shi, A. Sharma, T. Z. Zhao, C. Finn. *Waypoint-Based Imitation Learning for Robotic Manipulation*. CoRL 2023. https://arxiv.org/abs/2307.14326

T. Z. Zhao, V. Kumar, S. Levine, C. Finn. *Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware*. RSS 2023. https://arxiv.org/abs/2304.13705

S. Belkhale, Y. Cui, D. Sadigh. *HYDRA: Hybrid Robot Actions for Imitation Learning*. CoRL 2023. https://arxiv.org/abs/2306.17237

D. P. Kingma, J. Ba. *Adam: A Method for Stochastic Optimization*. ICLR 2015. https://arxiv.org/abs/1412.6980