Latent Space Planning Notes

Abstract

My notes on papers for latent space planning.

1 Efficient Planning in a Compact Latent Action Space

They introduce Trajectory Autoencoding Planning (TAP) for planning in the learned compact latent action space. Sequence of actions are mapped into a discrete latent space. This reduces the number of variables the model considers during planning.
TAP reasons about the trajectory as a whole because every latent represents L sequence of actions. This scales well with higher dimensions as the latent space is smaller and more structured than the raw action space.
The latent is learned using a state conditioned (gets the first state as input) VQ-VAE, so the latent space is discrete. Causal transformers are used for the encoder, decoder and the prior policy. The policy is trained to maximize the expected return (g) and uses Monte carlo Sampling to estimate the gradient.

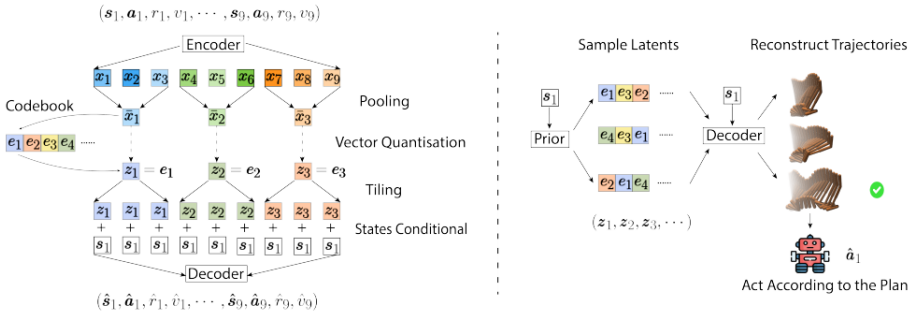


Figure 1:

2 Learning Predictive Representations for Deformable Objects Using Contrastive Estimation

In this paper they learn the latent dynamics of a deformable object (rope and cloth) using contrastive estimation (InfoNCE Loss). The latent model and the forward model are trained jointly.
Planning is done using a model predictive where several actions are sampled and are run through the forward model from the current $\mathbf{z}_t$. Then choose action theta produces $\hat{\mathbf{z}}_{t+1}$ closest to the goal embedding (l2-distance).
Contrastive forward models outperforms all baselines because they can generalize well as the latent space is task driven. If MSE Loss was used, reconstruction would be the priority and hence lighting and other sorts of noise will be fit, whereas, CFM regularizes to encode the latent space with task relevant information by not fitting the noise. Baselines like autoencoders and PlaNet fail to generalize effectively with DR, as they focus on reconstructing fine-grained details irrelevant to the task (e.g., lighting and color).
Domain randomization in sim facilitated Sim-to-Real.

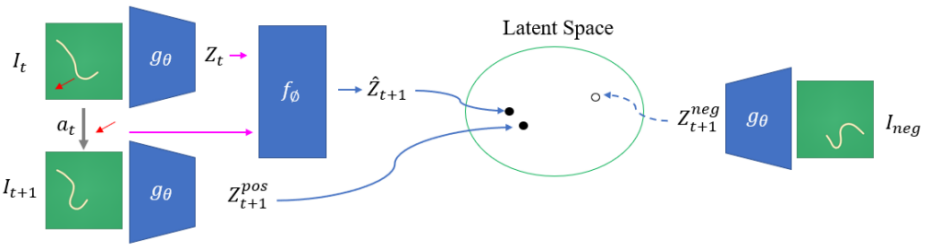


Figure 2: Overview of our contrastive forward model. Training data consists of (image, next image, action) tuples and we learn the encoder and forward model jointly. The contrastive loss objective brings the positive embedding pairs closer together and the negative embeddings further away.

Figure 2:

2.1 InfoNCE Loss

InfoNCE (Information Noise-Contrastive Estimation) Loss is used to learn representations by maximizing the mutual information between the input and the output. It contrasts the positive pairs (correct pairs) against negative pairs (incorrect pairs) to ensure that the learned representations are informative and discriminative. The loss is defined as:

$$\mathcal{L}_{\text{InfoNCE}} = -\mathbb{E}\left[\log \frac{h(\hat{\mathbf{z}}_{t+1}, \mathbf{z}_{t+1})}{\sum_{i=1}^{k} h(\hat{\mathbf{z}}_{t+1}, \tilde{\mathbf{z}}_i)}\right]$$

where $h(\cdot, \cdot)$ is a similarity function, $\hat{\mathbf{z}}_{t+1}$ and $\mathbf{z}_{t+1}$ are the latent representations of the positive pair, and $\tilde{\mathbf{z}}_i$ are the latent representations of the negative samples.

3 Physics-Informed Neural ODE (PINODE): Embedding Physics into Models using Collocation Points

$$\frac{d}{dt}x(t) = f(x(t)) \qquad z(T) = z(0) + \int_0^T h(z(t))\,dt$$

$$\frac{d}{dt}z(t) = h(z(t)) \qquad x(T) = \psi(z(T))$$

$$z(0) = \psi^{-1}(x(0))$$

The decoder, encoder are ψ, ϕ respectively. Latent dynamics evolve according to h.

$$\mathcal{L}^{\theta}_{\text{data}} = \frac{1}{2\sigma^2}\sum_{i=1}^{k}\left[\omega_1\sum_{j=1}^{p}\|x_i(t_j) - \psi_\theta(\phi_\theta(x_i(t_j)))\|^2 + \omega_2\sum_{j=1}^{p}\left\|\psi_\theta\left(\phi_\theta(x_i(t_1)) + \int_{t_1}^{t_j} h(z(t))\,dt\right) - x_i(t_j)\right\|^2\right]$$

$(\bar{x}, f(\bar{x}))$ are collocation points sampled from the space to incorporate physics through loss fn.

$$\frac{dz(x(t))}{dt} = h(\phi(x(t)))$$

$$\frac{dz(x(t))}{dt} = \frac{dz}{dx}\frac{dx}{dt} = \nabla\phi(x(t))^T f(x(t)) \qquad h(\phi(x(t))) = \nabla\phi(x)^T f(x)$$

$$\mathcal{L}^{\theta}_{\text{physics}} = \sum_{i=1}^{N}\left[\frac{\omega_3}{N}\|h_\theta(\phi_\theta(\bar{x}_i)) - \nabla\phi_\theta(\bar{x}_i)f(\bar{x}_i)\|^2 + \frac{\omega_4}{N}\|\bar{x}_i - \psi_\theta(\phi_\theta(\bar{x}_i))\|^2\right]$$

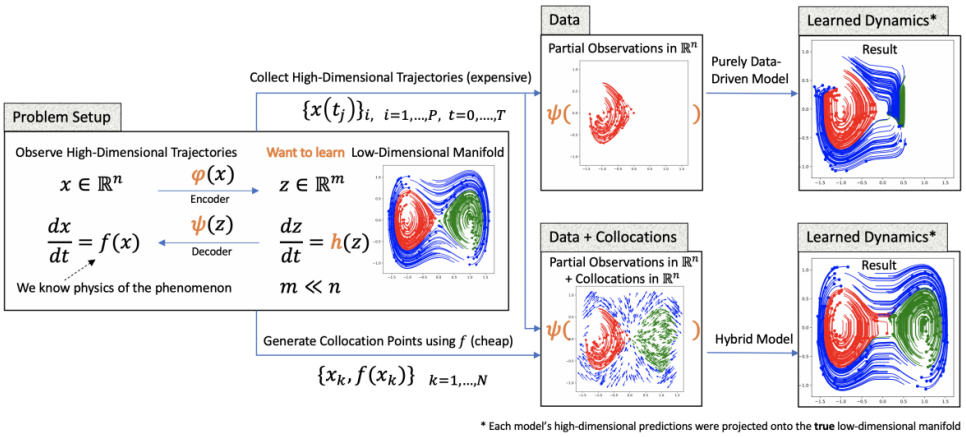


Figure 3:

4 Robot Motion Planning in Learned Latent Spaces

They have an autoencoder that converts x_t to z_t and a forward model that predicts z_{t+1} from z_t and a_t. A latent dynamics model that predicts z_{t+1} from z_t and u_t is trained. Lastly, a collision checking model is trained that takes in observations of any form, z_t and z_{t+1} to retrn either 0 or 1 indiciating if the latent states.
They sample points in the state space and create latent space samples. They then use those samples for RRT.

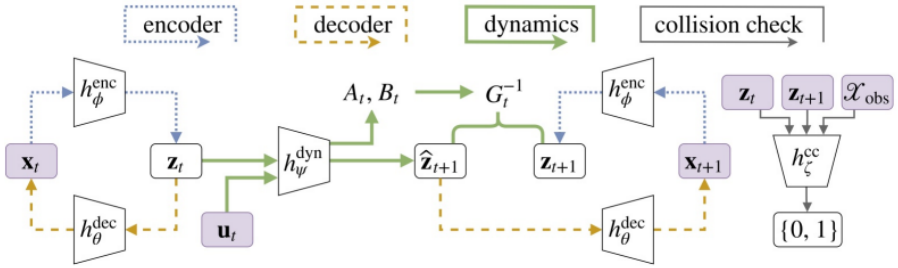


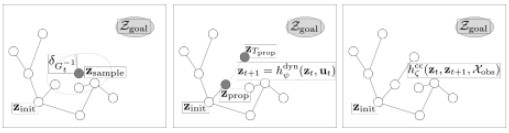
Figure 4:

1 Learned Latent RRT Outline

Parameters: $\delta_{G^{-1}}$, $T_{\max}$, $\mathcal{Z}_{\text{samples}}$, α
Inputs: $(\mathbf{x}_{\text{init}}, \mathcal{X}_{\text{goal}}, \mathcal{X}_{\text{free}})$

1 Encode problem input into latent space:
  $\mathbf{z}_{\text{init}} = h_\phi^{\text{enc}}(\mathbf{x}_{\text{init}})$, $\mathcal{Z}_{\text{goal}} = h_\phi^{\text{enc}}(\mathcal{X}_{\text{goal}})$
2 Add $\mathbf{z}_{\text{init}}$ to the tree $\mathcal{T}$
3 **while** time remains **do**
4 Sample $\mathbf{z}_{\text{sample}} \in \mathcal{Z}_{\text{samples}}$
5 $\mathbf{z}_{\text{prop}} = \texttt{BestFirstSelection}(\mathbf{z}_{\text{sample}}, \delta_{G_t^{-1}})$
6 Sample $T_{\text{prop}} \in \{1, \ldots, T_{\max}\}$ and $\mathbf{u}_{0,\ldots,T_{\text{prop}}-1} \in \mathcal{U}$
7 Propagate dynamics $\mathbf{z}_{t+1} = h_\psi^{\text{dyn}}(\mathbf{z}_t, \mathbf{u}_t)$ from $\mathbf{z}_0 = \mathbf{z}_{\text{prop}}$ for T_{prop} time steps, to generate latent trajectory $(\mathbf{z}_0, \mathbf{u}_0, \ldots, \mathbf{z}_{T_{\text{prop}}-1}, \mathbf{u}_{T_{\text{prop}}-1}, \mathbf{z}_{T_{\text{prop}}}, 0)$
8 **if** $\text{sigmoid}(h_\zeta^{\text{cc}}(\mathbf{z}_t, \mathbf{z}_{t+1}, \mathcal{X}_{\text{obs}})) > \alpha \, \forall \, t$ **then**
9 Add $\mathbf{z}_T$ to the tree $\mathcal{T}$
10 **end if**
11 **end while**
12 **if** $\mathcal{T} \cap \mathcal{Z}_{\text{goal}} = \emptyset$ **then**
13 Report failure
14 **else**
15 Select lowest-cost trajectory with its end in $\mathcal{T} \cap \mathcal{Z}_{\text{goal}}$
16 Project back to full state space through h_θ^{dec} and return $(\mathbf{x}_{\text{init}}, \mathbf{u}_0, \ldots, \mathbf{x}_T, \mathbf{u}_T)$, where T is time steps to $\mathbf{z}_{\text{goal}}$
17 **end if**

Figure 5:



(a) Lines 4-5 (b) Lines 6-7 (c) Lines 8-9

Figure 6:

5 Learning Latent Dynamics for Planning from Pixels

They propose a method to learn latent dynamics directly from pixel observations. The method involves training a latent dynamics model that can predict future latent states from current latent states and actions. This approach allows for planning directly in the latent space, which is more compact and structured than the raw pixel space.
The latent dynamics model is trained using a combination of reconstruction loss and a dynamics loss. The reconstruction loss ensures that the latent space captures the essential information from the pixel observations, while the dynamics loss ensures that the latent space captures the temporal dynamics of the system.

Algorithm 1: Deep Planning Network (PlaNet)

Input :

R	Action repeat	$p(s_t \mid s_{t-1}, a_{t-1})$	Transition model
S	Seed episodes	$p(o_t \mid s_t)$	Observation model
C	Collect interval	$p(r_t \mid s_t)$	Reward model
B	Batch size	$q(s_t \mid o_{\leq t}, a_{<t})$	Encoder
L	Chunk length	$p(\epsilon)$	Exploration noise
α	Learning rate		

Initialize dataset $\mathcal{D}$ with S random seed episodes.
Initialize model parameters θ randomly.
while *not converged* **do**
  `// Model fitting`
  for *update step* $s = 1..C$ **do**
    Draw sequence chunks $\{(o_t, a_t, r_t)_{t=k}^{L+k}\}_{i=1}^{B} \sim \mathcal{D}$ uniformly at random from the dataset.
    Compute loss $\mathcal{L}(\theta)$ from Equation 3.
    Update model parameters $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$.
  `// Data collection`
  $o_1 \leftarrow$ `env.reset()`
  for *time step* $t = 1..\lceil \frac{T}{R} \rceil$ **do**
    Infer belief over current state $q(s_t \mid o_{\leq t}, a_{<t})$ from the history.
    $a_t \leftarrow$ `planner`$(q(s_t \mid o_{\leq t}, a_{<t}), p)$, see Algorithm 2 in the appendix for details.
    Add exploration noise $\epsilon \sim p(\epsilon)$ to the action.
    for *action repeat* $k = 1..R$ **do**
      $r_t^k, o_{t+1}^k \leftarrow$ `env.step`(a_t)
    $r_t, o_{t+1} \leftarrow \sum_{k=1}^{R} r_t^k, o_{t+1}^R$
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(o_t, a_t, r_t)_{t=1}^{T}\}$

Figure 7: