

Supplemental Material

1. Details of the proposed method

Here we describe the implementation details of our proposed *visual posturing* controller. The controller was implemented using Tensorflow 2.8.0.

1.1. World model and learning

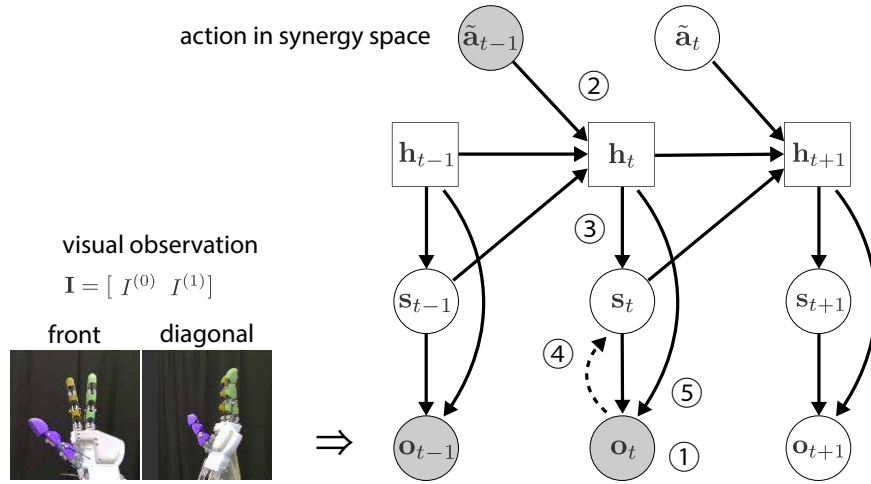


Figure 1. Recurrent state-space model

1.1.1. Observation and embedding

The observation of the world model (Fig. 1 ①) was a set of two images; a front view and a diagonal view. An image of each view was RGB of 128×128 pixels. Each pixel was an unsigned integer within $[0, 255]$. The images were first normalized to $[-0.5, 0.5]$. Subsequently, they were masked to remove the background. Masks were generated using depth images obtained from the depth cameras. Pixels were set to 1.0 if they were within a specific distance range and 0.0 otherwise. The ranges were determined so that only the robotic hands were extracted.

The masked images were fed into a CNN image encoder. Images from the two views were processed with separate CNNs. Each CNN consisted of five convolution layers (filter numbers: 4, 8, 16, 32, 64, kernel size: 4×4 , strides: 2×2 , activation: ReLU). The outputs of the CNNs were flattened and concatenated and fed into a fully connected (FC) layer to calculate an observation embedding \mathbf{e}_t , which had 1024 elements.

1.1.2. Updating deterministic term

A deterministic term was calculated from the previous latent state ($\mathbf{z}_{t-1} = [\mathbf{h}_{t-1}; \mathbf{s}_{t-1}]$) and the previous action in the synergy space ($\tilde{\mathbf{a}}_{t-1}$) (Fig. 1 ②). Specifically, \mathbf{s}_{t-1} and $\tilde{\mathbf{a}}_{t-1}$ were first concatenated and fed into a FC layer (activation functions for FC layers were exponential linear unit (ELU) unless otherwise mentioned) to calculate a vector of 200 elements. This vector was fed into a GRU (`tensorflow.keras.layers.GRU`), whose hidden layer had 100 units. The current hidden layer of the GRU was used as \mathbf{h}_t ; hence, \mathbf{h}_t was a vector of 100 elements.

1.1.3. Prior prediction

A prior distribution p^{prior} of the stochastic term \mathbf{s}_t was calculated solely from a deterministic term \mathbf{h}_t (Fig. 1 ③). \mathbf{h}_t was fed into FC layers to calculate a vector of 200 elements. Subsequently, it was fed into two different FC layers (without activation functions) to calculate $\boldsymbol{\mu}_s^{\text{pri}}$ and $\boldsymbol{\sigma}_s^{\text{pri}}$, which were vectors of 100 elements. The second vector $\boldsymbol{\sigma}_s^{\text{pri}}$ was post processed as $\boldsymbol{\sigma}_s^{\text{pri}} \leftarrow \text{softplus}(\boldsymbol{\sigma}_s^{\text{pri}}) + 0.1$. A prior distribution was instantiated as a multivariate normal distribution whose mean was $\boldsymbol{\mu}_s^{\text{pri}}$ and diagonal standard deviation was $\boldsymbol{\sigma}_s^{\text{pri}}$. \mathbf{s}_t was sampled from this prior for imagination during planning.

1.1.4. Posterior inference

A posterior distribution $p^{\text{posterior}}$ of the stochastic term \mathbf{s}_t was calculated from a deterministic term \mathbf{h}_t and an observation embedding \mathbf{e}_t (Fig. 1 ④). First, \mathbf{h}_t and \mathbf{e}_t were concatenated, and they were fed into an FC layer to calculate a vector of 200 elements. Subsequently, it was fed into two different FC layers (without activation functions) to calculate $\boldsymbol{\mu}_s^{\text{post}}$ and $\boldsymbol{\sigma}_s^{\text{post}}$, which were vectors of 100 elements. The second vector $\boldsymbol{\sigma}_s^{\text{post}}$ was post processed as $\boldsymbol{\sigma}_s^{\text{post}} \leftarrow \text{softplus}(\boldsymbol{\sigma}_s^{\text{post}}) + 0.1$. A posterior distribution was instantiated as a multivariate normal distribution whose mean was $\boldsymbol{\mu}_s^{\text{post}}$ and diagonal standard deviation was $\boldsymbol{\sigma}_s^{\text{post}}$. \mathbf{s}_t was sampled from this posterior to update the current belief of the system state after observation at each step.

1.1.5. Reconstruction of observation

An observation \mathbf{o}_t was reconstructed from \mathbf{h}_t and \mathbf{s}_t (Fig. 1 ⑤). First, a latent state $\mathbf{z}_t = [\mathbf{h}_t; \mathbf{s}_t]$ was fed into an FC layer (no activation functions) to calculate a vector of 512 elements. Subsequently, it was fed into two independent stacks of transposed convolution layers to generate two tensors \hat{I}_0^{mean} and \hat{I}_1^{mean} , whose size were $128 \times 128 \times 3$. A stack of transposed convolution layers (`tensorflow.keras.layers.Conv2DTranspose`) had five layers (number of filters: 64, 32, 14, 8, 3, kernel size: 5×5 , stride: 2×2). \hat{I}_0^{mean} and \hat{I}_1^{mean} were stacked to compose a reconstruction $\mathbf{I}_t^{\text{mean}}$. A generative distribution p^{gen} was instantiated as a multivariate normal distribution whose mean was $\mathbf{I}_t^{\text{mean}}$ and the standard deviation was 1.0. This distribution was used to calculate the likelihood in the loss function.

1.1.6. Updating world model

During training, episodes were stored in a buffer (either D_{random} or D_{replay}). B episodes were sampled uniformly at random from $D_{\text{random}} \cup D_{\text{replay}}$ to update the world model. From each sampled episode, a sequence of L steps was randomly sampled. As a result,

a batch of $B \times L$ samples were generated for one update iteration. In this study, we set $B = 50$ and $L = 21$.

$$L_{\text{RSSM}} = L_{\text{likelihood}} + D_{\text{KL}} \quad (1)$$

$$L_{\text{likelihood}} = \mathbb{E}_{\mathbf{o}_t \sim D_{\text{replay}}, \mathbf{z}_t \sim p_{\text{post}}} [\log p^{\text{gen}}(\mathbf{o}_t | \mathbf{z}_t)] \quad (2)$$

$$D_{\text{KL}} = \mathbb{E}_{\mathbf{z}_{t-1}} [D_{\text{KL}}(p^{\text{post}}(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{a}_{t-1}, \mathbf{o}_t) \parallel p^{\text{prior}}(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{a}_{t-1}))] \quad (3)$$

The likelihood term (2) was calculated as follows. First, latent states for each sequence in a batch were calculated sequentially, starting from the first observation and action. The initial hidden state of the GRU was set to zero, and latent states were calculated step by step by feeding in the next observation and action. After inferring all the latent states, generative distribution p^{gen} was calculated from the states following the procedure described in Section 1.1.5. The log-likelihood of observations provided the generative distribution was averaged over the batch to obtain (2). The KL divergence term (3) was calculated by averaging KL divergence between the posterior and the prior of the stochastic variables in the batch.

All the parameters in the world model were updated by minimizing (1) using SGD. We used Adam optimizer (`tensorflow.optimizers.Adam`) with a learning rate 6.0×10^{-4} . We clipped the norm of gradients to 100.0 for learning stability.

1.2. Model predictive path integral

Algorithm 1 Action sequence planning at step t

Input: latent state \mathbf{z}_t , goal latent state \mathbf{z}_g , plan horizon H

latent dynamics p , previous momentum $\mathbf{m}_{t-1:t+H-2}$

previous action sequence $\boldsymbol{\zeta}_{t-1:t+H-2}$

Output: $\mathbf{m}_{t:t+H-1}$, $\boldsymbol{\zeta}_{t:t+H-1}$

$\boldsymbol{\zeta}_{t:t+H-1} \leftarrow [\boldsymbol{\zeta}_{t:t+H-2}; \mathbf{0}]$

$\mathbf{m}_{t:t+H-1} \leftarrow [\mathbf{m}_{t:t+H-2}; \mathbf{0}]$

for $j = 1 \dots M$ **do**

for $k = 1 \dots K$ **do**

$\boldsymbol{\epsilon}^{(k)} \leftarrow \mathcal{N}(\mathbf{0}, \delta^2 I)$ (sample noise)

$\tilde{\mathbf{a}}_{t:t+H-1}^{(k)} \leftarrow \boldsymbol{\zeta}_{t:t+H-1} + \boldsymbol{\epsilon}^{(k)} + \gamma \mathbf{m}_{t:t+H-1}$ (generate action samples)

$\mathbf{z} \leftarrow \mathbf{z}_t$

for $\tau = 0 \dots H-1$ **do**

$\mathbf{z} \sim p(\mathbf{z}, \tilde{\mathbf{a}}_{\tau}^{(k)})$ (imagination)

end for

$d^{(k)} \leftarrow \|\mathbf{z}_g - \mathbf{z}\|_2$ (calculate distance from the goal)

end for

$\Delta \leftarrow \left(\sum_{k=1}^K \exp(-d^{(k)} \lambda^{-1}) \boldsymbol{\epsilon}^{(k)} \right) / \left(\sum_{k=1}^K \exp(-d^{(k)} \lambda^{-1}) \right)$

$\mathbf{m}_{t:t+H-1} \leftarrow \gamma \mathbf{m}_{t:t+H-1} + \Delta$ (NAG)

$\boldsymbol{\zeta}_{t:t+H-1} \leftarrow \boldsymbol{\zeta}_{t:t+H-1} + \mathbf{m}_{t:t+H-1}$

end for

Algorithm 1 is the pseudo-code for MPPI in the world model. Nesterov Accelerated

Gradient (NAG) was adopted to accelerate the planning. We set the following parameters: $M = 40$, $K = 250$, $\delta = 0.05$, $\lambda = 10.0$, and $\gamma = 0.8$. Planning horizon H was set to $\max(\tilde{H}, T - t)$, where $\tilde{H} = 10$. At the beginning of an episode, where there was no preceding action sequence, ζ was set to a zero vector.

2. Hardware details

2.1. Hand design and fabrication

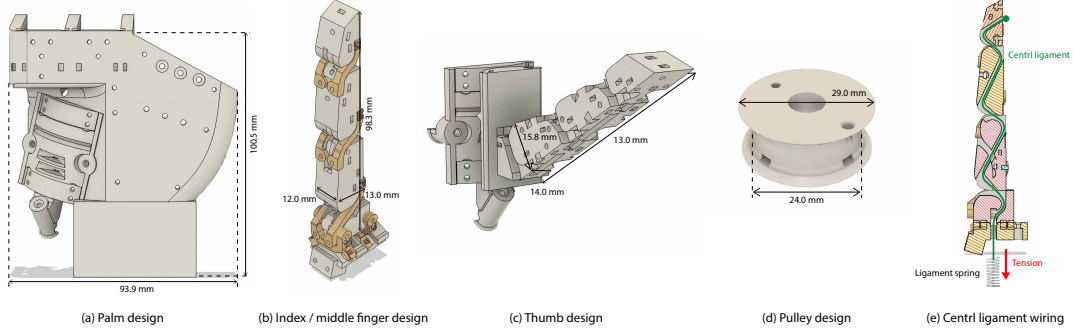


Figure 2. Hardware designs and dimensions. (a) Palm design. (b) Finger design for index and middle fingers. This part was mounted on the palm. The metacarpophalangeal (MCP) consists of two rolling contact joints. (c) Finger design for the thumb. The carpometacarpal (CMC) consists of two rolling contact joints. (d) Pulley design. It was fixed to a servo motor shaft. A tendon was fixed to this pulley using a screw. (e) Illustration of central ligament wiring for the index finger. The central ligament string was fixed to a spring to generate tension.

The design of a physical tendon-driven manipulator for the experiment is presented in Fig. 2. We designed the hardware using Fusion360 (from Autodesk). Most of the parts were made of VeroClear (translucent) or VeroWhitePlus (white color). They were printed using Objet260 Connex3 (from Stratasys). Ligaments (amber parts in Fig. 2) were made of Flexa Bright(TPU, A Shore hardness: 79) and printed using Lisa-Pro (from Sinterit). There was a central ligament inside each finger, as shown in the cross-section view in Fig. 2(e). It was fixed to the palm via a ligament spring to extend the finger. Photos of the fabricated hand was presented in Fig. 3. Fingers were covered by colored skins (Flexa Bright) shown in Fig. 3(a). They were fixed to fingers with screws. We used Spectra Dyneema ropes of diameter 1.5mm for central ligaments and tendons (the black ropes in Fig. 3). Tendons were guided to motors through polytetrafluoroethylene tubes (the white tubes shown in Fig. 3).

2.2. Tension sensor

Each tendon was equipped with a tension sensor to avoid extremely high tension (Fig. 4). When a pulley pulled a unit, a slider pushed a pressure sensor FSR400 (Interlink Electronics Inc). When a sensor detected tension over a threshold, the motion of the associated motor was restricted to avoid additional tension. In the experiment, we restricted the motion of the associated motor when the resistance of an FSR400 fell below $1.92 \text{ k}\Omega$, in order to limit the tension.

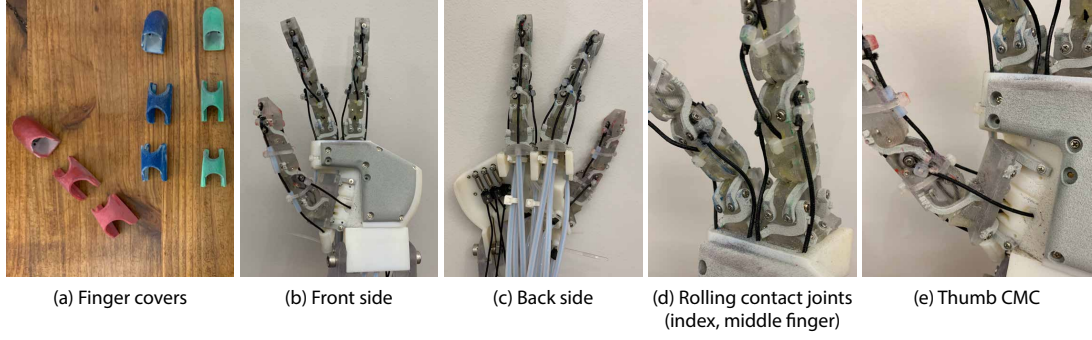


Figure 3. Images of the fabricated hand.

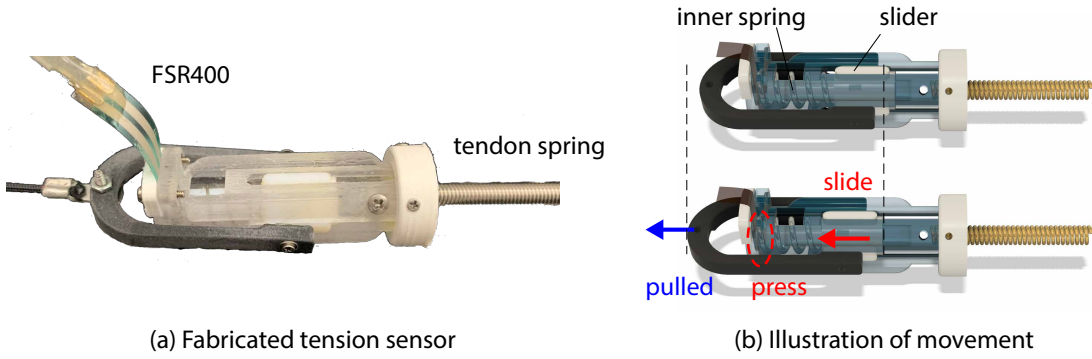


Figure 4. Tension sensor

3. Collection of typical hand postures

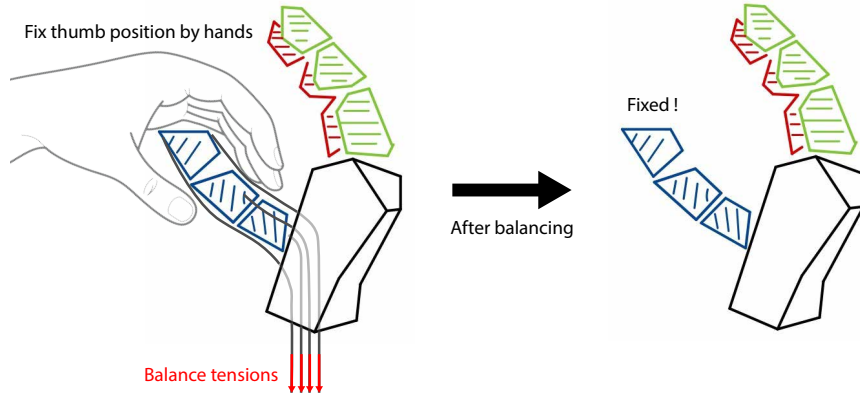


Figure 5. An illustration of typical posture collection where a posture for the thumb is being collected.

Typical postures were collected for each finger separately. Figure 5 illustrates a procedure to collect a typical posture of the thumb. As depicted in the figure, the finger was fixed at a target position using the experimenter's hands. After that, the angles of motors that were associated with the finger were adjusted iteratively to achieve a specific tension. Tensions were monitored by the tension sensors described in Sec. 2.1. Once the tensions were balanced, the finger was fixed at the position without the

external support. The motor angles of this state were recorded for synergy extraction. We collected 14 postures for the thumb, 12 postures for the index finger, and 12 postures for the middle finger, respectively. Figure 8 shows all the typical postures collected for the experiment.

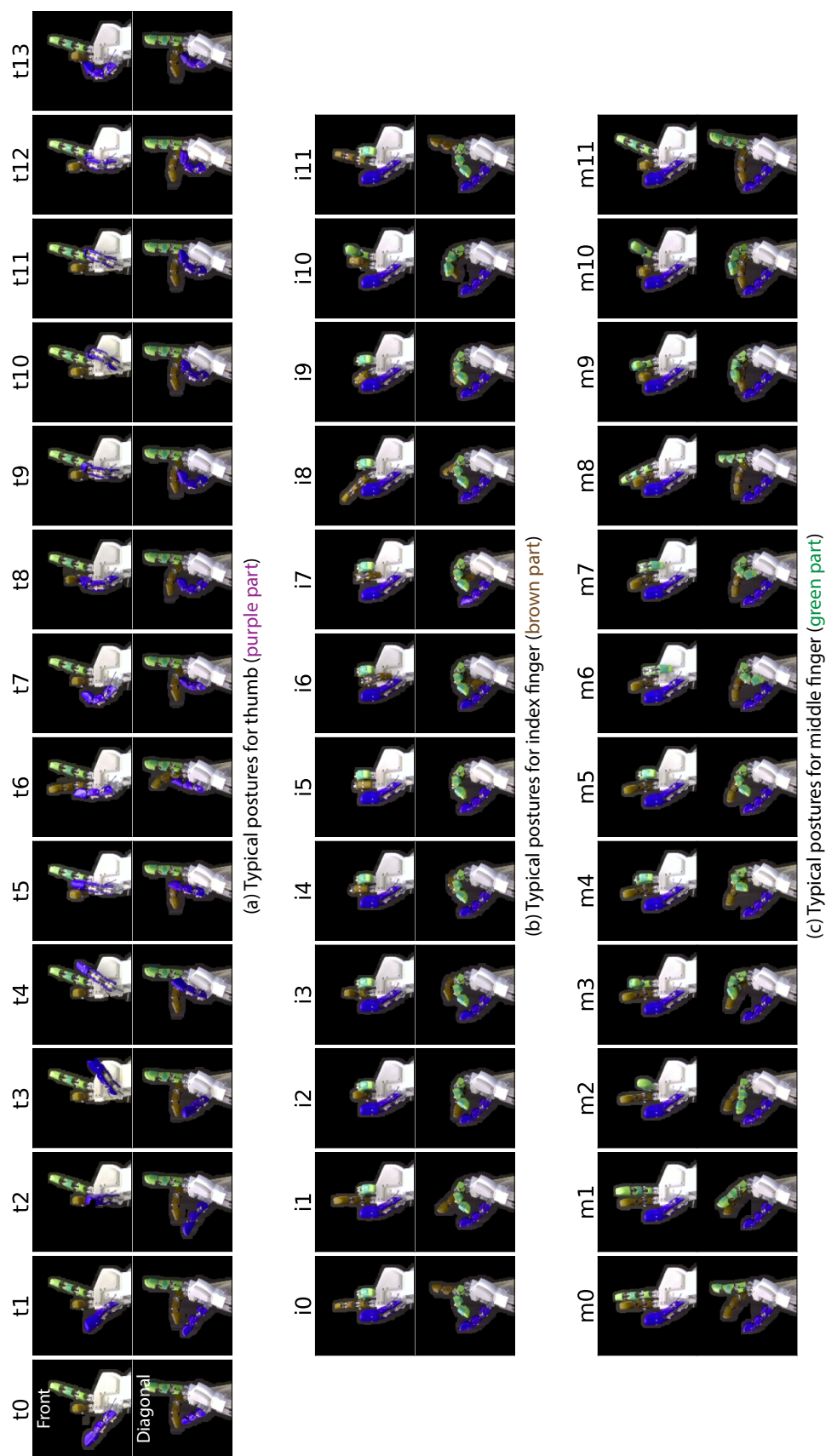


Figure 6. List of collected typical hand postures.

4. Baseline implementations

4.1. regression

A regression model used the same CNNs described in 1.1.1 to generate an observation embedding. An embedding was fed into two FC layers (400 units) to calculate a vector of 400 elements. Subsequently, it was fed into two separate FC layers (no activation functions) to calculate a mean vector \mathbf{v}_{mean} and a standard deviation vector \mathbf{v}_{std} . These vectors were post-processed as follows:

$$\mathbf{v}_{\text{mean}} \leftarrow 5.0 \times \tanh\left(\frac{\mathbf{v}_{\text{mean}}}{5.0}\right) \quad (4)$$

$$\mathbf{v}_{\text{std}} \leftarrow \text{softplus}(\mathbf{v}_{\text{std}} + \log(e^{5.0} - 1.0)) + 0.0001 \quad (5)$$

Action distribution p^{regress} was first instantiated as a multivariate normal distribution, whose mean was \mathbf{v}_{mean} and variance was $(\mathbf{v}_{\text{std}})^T I \mathbf{v}_{\text{std}}$. Then, its sample space was transformed to the interval $[-1.0, 1.0]$ using a tanh bijection function.

During training, pairs of an image observation and a corresponding set of motor angles were sampled from $D_{\text{random}} \cup D_{\text{replay}}$ as a mini-batch. The following loss function was minimized using the mini-batch:

$$L_{\text{regress}} = \mathbb{E}_{\mathbf{o}_t, \mathbf{a}_t \sim D_{\text{random}} \cup D_{\text{replay}}} [-\log p^{\text{regress}}(\text{SE}(\mathbf{a}_t) | \mathbf{o}_t)] \quad (6)$$

Optimization was conducted using SGD with an Adam optimizer (learning rate being 6.0×10^{-4} , gradient norm clipped to 100.0). The number of updates in both pretraining and reinforcement learning were aligned with the proposed method.

4.2. regression(r)

10k random samples were collected for this baseline, and a model was trained only using this random data (i.e., only the pretraining phase was conducted and no reinforcement learning phase). The pretraining phase lasted until conducting 10k updates.

4.3. without synergies

The synergy module in the proposed method was replaced with a fixed linear transformation to fit the action range into $[-1.0, 1.0]$ in the world model.

$$\tilde{\mathbf{a}}_t = \frac{\mathbf{a}_t - \mathbf{a}_{\min}}{\mathbf{a}_{\max} - \mathbf{a}_{\min}} \times 2.0 - 1.0 \quad (7)$$

$$\mathbf{a}_t = 0.5 \times (\tilde{\mathbf{a}}_t + 1.0) \times (\mathbf{a}_{\max} - \mathbf{a}_{\min}) + \mathbf{a}_{\min} \quad (8)$$

4.4. Dreamer

We based the implementation of this baseline on the official one (<https://github.com/danijar/dreamer>). Modifications to the implementation were described here.

An actor module was an FC network with four layers (400 units) followed by two separate FC layers (no activation functions) that calculated the mean and standard deviation of a normal distribution. Then, its sampling space was converted to the

interval $[-1.0, 1.0]$, as described in Sec. 4.1. The module received the current latent state \mathbf{z}_t and a latent goal \mathbf{z}_g , and concatenated them before feeding to the network.

A value module was an FC network with three layers (400 units) followed by a single FC layer (no activation functions) that calculated the mean of a normal distribution. A standard deviation was fixed at 1.0. The module received the current latent state \mathbf{z}_t and a latent goal \mathbf{z}_g , and concatenated them before feeding to the network.

A reward module was removed from the original implementation, and Euclidean distance between \mathbf{z}_t and \mathbf{z}_g was used to evaluate imagined trajectories. A latent state consisted of a deterministic term \mathbf{h}_t (100 elements) and a stochastic term \mathbf{s}_t (30 elements).

5. Posturing Behaviors

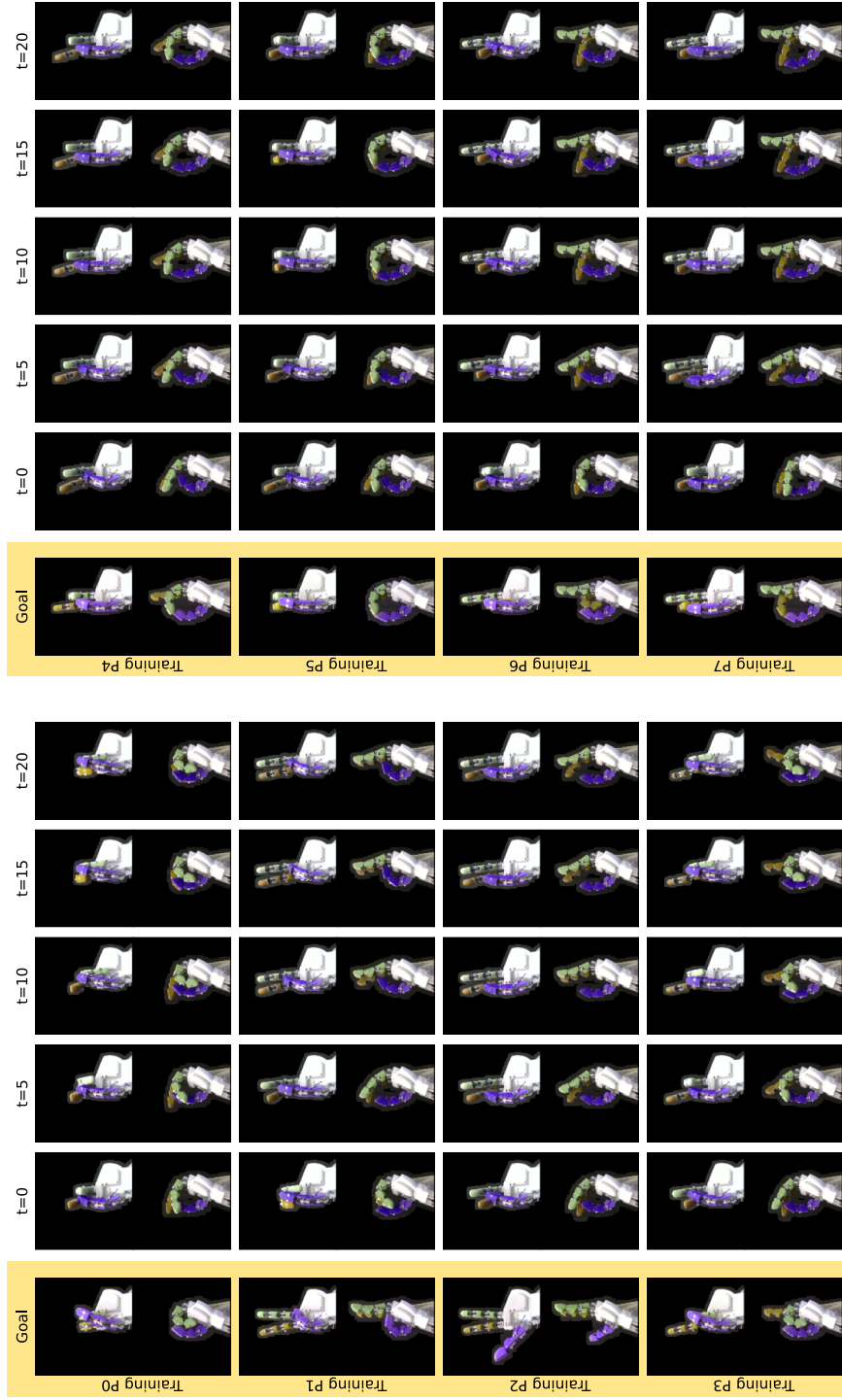


Figure 7. Timelapse sequences of hand posturing episodes achieved using the proposed method. Goal hand postures are from the training set.

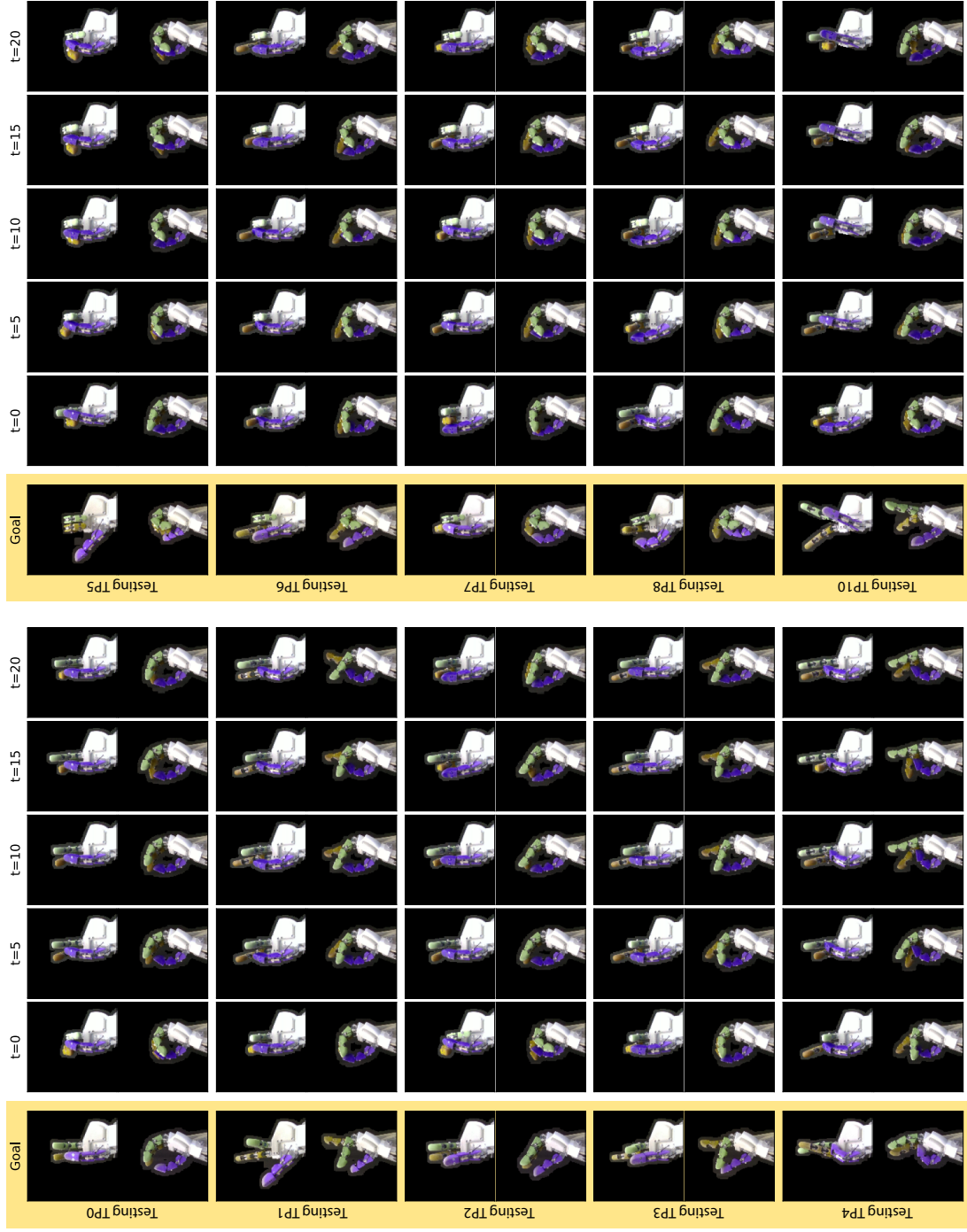


Figure 8. Timestep sequences of hand posturing episodes achieved using the proposed method. Goal hand postures are from the testing set.