TACTIC: Tactile and Vision Conditioned Contact-Centric Control for Whole-Arm Manipulation

tactic-2026.github.io

Supplementary Material

Author Names Omitted for Anonymous Review
Paper-ID [757]

Table of Contents

A. System Overview and Latency Breakdown

A. *Pipeline overview*

At each MPC step t, the system runs the following sequence: **Sensor inputs** $\rightarrow$ **Mask generation** $\rightarrow$ **Contact-centric observation** $\rightarrow$ **Encoding** $\rightarrow$ **Rollouts (draft + kinematic + full) + decoding** $\rightarrow$ **Scoring** $\rightarrow$ **Execution**. We acquire synchronized sensor data at 40Hz. We then generate the proximity mask M_t (segmentation + kd-tree queries + taxel projection) which takes 5-8ms. We assemble the contact-centric observation o_t^{cc} and encode it into a latent $\mathbf{z}_t = \phi(o_t^{\mathrm{cc}})$, updating the rolling context $\mathbf{z}_{t-L+1:t}$ ($\sim$6ms). After receiving the sensor observation, in parallel, MPPI samples N action sequences and applies contact-aware shaping. For each candidate sequence, we roll out analytical kinematics $\mathbf{F}_k$ to obtain kinematic signals $\hat{s}^{(j)}_{t+1:t+H}$ and roll out the draft latent predictor $\mathbf{F}_l^{\mathrm{draft}}$ for all particles to obtain predicted latents; these two rollouts are computed in parallel. We then select the top-M candidates by draft score and re-score only those with the full predictor ($\sim$65-70ms), producing the final costs used to compute MPPI weights and update the nominal sequence. Finally, we execute the first m actions $\mathbf{u}^{\star}_{t:t+m-1}$ using a low-level controller running at 1000 Hz (joint-space compliant controller), and replan at $t+m$ with updated observations. Overall, our MPC runs at close to 12Hz.

B. *Hyperparameter table*

TABLE I: Hyperparameters for TACTIC.

Parameter	Value
MPC horizon H	`8`
MPC time step Δt	`0.025`
MPPI particles N	`100`
Two-fidelity top-M	`10`
Latent history length L	`3`
Control mode	`joint-space impedance`

B. Contact-Centric Representation Details

A. *Proximity mask construction*

We assume RGB and depth are spatially aligned and obtain camera intrinsics $\mathbf{K}$ using `pyrealsense` API. We use a fixed extrinsic transform between the robot base and camera, which is needed only for projecting taxels into the image. We use the `easy_handeye` [] package to obtain the extrinsics. For each pixel (u,v) with valid depth $z(u,v)$, we unproject to 3D in the camera frame as $\mathbf{p}(u,v) = z\,\mathbf{K}^{-1}[u,\ v,\ 1]^{\top}$. We form $P_r = \{\mathbf{p}(u,v) \mid (u,v) \in \Omega_r\}$ and $P_h = \{\mathbf{p}(u,v) \mid (u,v) \in \Omega_h\}$ from robot/object masks. The object masks are obtained using YOLO11n-seg. To train this model, we semi-automatically annotate the robot, maze, and manikin class in our dataset using SAM2.1. After training, we convert the model to TensorRT and run the model with an inference time of 2-3ms. We build a kd-tree on P_h and query each $\mathbf{p} \in P_r$ to compute $d(\mathbf{p}) = \min_{\mathbf{p}_h \in P_h} \|\mathbf{p} - \mathbf{p}_h\|_2$. We convert distances to an image-aligned proximity field by applying the following linear clipping:

$$M_t^{\mathrm{vis}}(u,v) = \mathrm{clip}\left(1 - \frac{d(\mathbf{p}(u,v))}{d_{\max}},\ 0,\ 1\right),$$

where $d_{\max}$ is the maximum distance considered relevant for anticipating contact. Pixels outside Ω_r are set to zero. We render a separate blank mask M_t^{tax} (zeros, same resolution as the image). For each taxel k with measured force exceeding $f_{\min}$, we compute its 3D pose via forward kinematics, transform it into the camera frame, project it with $\mathbf{K}$, and draw a filled disk of radius r pixels centered at the projected location.We fuse the visual proximity field and taxel indicator by per-pixel maximum and clip to $[0,1]$:

$$M_t(u,v) = \max\big(M_t^{\mathrm{vis}}(u,v),\ M_t^{\mathrm{tax}}(u,v)\big)\,.$$

In all experiments, we use $d_{\max} = 0.05\,\mathrm{m}$, $f_{\min} = 2\,\mathrm{N}$, and $r = 5$px.

B. *Encoder implementation details*

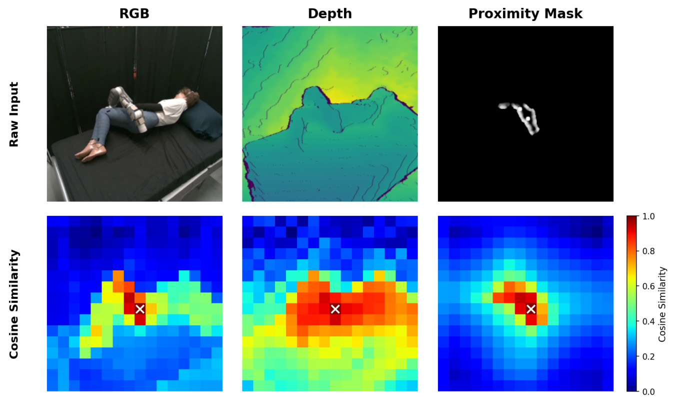

Fig. 1: Cosine similarity for RGB, Depth, and Proximity Mask.

Vision backbone. We apply the same DINOv3 backbone to I_t, D_t, and M_t in a batched forward pass. Depth and mask inputs are replicated to three channels and normalized using the same preprocessing as RGB (after scaling depth to metric units). The DINO feature map is pooled to a fixed-dimensional vector and projected to the vision embedding used by the world model. As shown in Fig. 1, the cosine similarity maps show that the encoder is able to extract spatially meaningful features from the inputs.

C. Contact-Aware Action Sampling Details

A. *Active contact set and contact geometry*

We define the active contact set $\mathcal{K}_t$ from taxels whose measured normal force exceeds threshold $f_{\min}$. For each active taxel k, we treat the contact point as the taxel center obtained from forward kinematics. The corresponding translational Jacobian $\mathbf{J}_{k,t} \in \mathbb{R}^{3\times n}$ is computed from the robot model Each taxel has a known outward surface normal in link frame. We transform it to the world frame to obtain a candidate $\mathbf{n}_{k,t}$.

B. Force weighting and projection operators

We use the taxel force magnitude $f_{k,t}$ (scalar normal force) and define $\rho_{k,t} = \mathrm{sat}(f_{k,t}/F_{\mathrm{sat}})$ to reduce the influence of weak/noisy contacts while preventing a single contact from dominating via saturation. We compute $\mathbf{G}_t^{\mathrm{force}}$ using the small matrix inverse $(\widetilde{\mathbf{J}^n}_t \widetilde{\mathbf{J}^n}_t^\top + \varepsilon I)^{-1}$ with a stable factorization (e.g., Cholesky). When $K_t = 0$, we set $\mathbf{G}_t^{\mathrm{force}} = \mathbf{0}$ and $\mathbf{G}_t^{\mathrm{null}} = I$. The ridge term $\varepsilon > 0$ prevents numerical issues when contact normals are redundant (e.g., nearly collinear rows in $\widetilde{\mathbf{J}^n}_t$).

Parameter	Value
σ_{force}	1.0
σ_{null}	0.7
σ_{min}	0.1
F_{sat}	30.0 N
ε	10^{-4}
F_{safe}	20.0 N
Max safe contacts K	≤ 6

TABLE II: Hyperparameter values for contact-conditioned shaping and safety projection.

C. Safety projection QP

We form $\mathcal{K}_t^{\mathrm{safe}} = \{k \in \mathcal{K}_t \mid f_{k,t} \geq F_{\mathrm{safe}}\}$ and stack the corresponding rows into $\mathbf{J}_t^{\mathrm{safe}}$. Because $|\mathcal{K}_t^{\mathrm{safe}}|$ is small, we solve it efficiently with active-set method. All 2^K active sets are enumerated, the corresponding equality-constrained least-squares problems are solved in closed form, and the feasible solution with minimum objective is selected. If $\mathcal{K}_t^{\mathrm{safe}} = \emptyset$, we skip the projection.

Table II shows the tuned parameters obtained in simulation by commanding the robot to regulate a fixed goal end-effector pose while injecting contacts of varying magnitudes and locations along the arm. We selected values that minimized safety violations and improved recovery behavior, using metrics including contact-limit violations, escape (separation) speed under high force, and tracking error.

The projection assumes contact normals are locally constant over the short MPC horizon, which is reasonable under our planning frequency. Since the constraint is applied only to contacts above F_{safe} and enforces separation rather than precise force control, errors in the assumed normals result in conservative behavior rather than unsafe actions.

Metric	Contact-aware	Contact-aware + QP
Violations	84.3 ± 39.8	**50.0 ± 20.1**

TABLE III: Ablation results for QP-based projection.

We perform multi-contact ablation experiments to observe the effect of QP-based projection. Table III reports the number of safety force violations under contact-aware control, comparing contact-conditioned shap alone against shaping combined with our QP-based safety projection.

D. Hybrid Predictive Model and Planning Details

Each timestep is encoded into a latent vector $\mathbf{z}_t \in \mathbb{R}^{d_z}$ with $d_z = 384$. We form $\mathbf{z}_t$ by concatenation of modality embeddings from RGB, depth, proximity mask, forces, and proprioception. The predictor conditions on a context window $\mathbf{z}_{t-L+1:t}$ with $L = 3$. We implement $\mathbf{F}_l$ as a Transformer/Vision-Transformer-style predictor with 6 blocks, hidden size 384, and 8 attention heads. We use a frame-causal attention mask so that tokens from timestep t' attend only to timesteps $\leq t'$ within the context window. The predictor outputs $\hat{\mathbf{z}}_{t+1}$, and we obtain $\hat{\mathbf{z}}_{t+1:t+H}$ via iterative rollout.

Actions are commanded joint velocities $\mathbf{u}_t \in \mathbb{R}^n$. We normalize actions using per-dimension statistics (mean/std) and embed them with an MLP action encoder $\phi_{\mathrm{act}} : \mathbb{R}^n \to \mathbb{R}^{d_a}$ with $d_a = 10$ and architecture 1D Conv (kernel=1, stride=1).

When predicting $\hat{\mathbf{z}}_{t+k}$, we condition on the step-aligned action $\mathbf{u}_{t+k-1}$ by injecting $\phi_{\mathrm{act}}(\mathbf{u})$ into each Transformer block via AdaLN, i.e., the action embedding produces per-block scale/shift parameters.

Teacher-forced training uses the ground-truth context $\mathbf{z}_{t-L+1:t}$ to predict $\hat{\mathbf{z}}_{t+1:t+H}$. For the multistep rollout loss, we unroll autoregressively for k steps by updating the context with predicted latents using a sliding window: append $\hat{\mathbf{z}}$ and drop oldest. We use $K_{\mathrm{roll}} = 2$ and weights $\lambda_k = 0.25$.

We attach lightweight decoder heads ψ_m to $\hat{\mathbf{z}}_{t+i}$ for planning-relevant signals. The force decoder ψ_{force} is a 2-layer MLP with hidden dims [64, 32] and ReLU activations that predicts $\hat{\mathbf{f}}_{t+i} \in \mathbb{R}^{N_s}$; we train it with an ℓ_2 loss and use linear (unbounded) output to enforce no constraints (raw force prediction). For contact classification, we predict per-taxel contact probabilities with a sigmoid head and supervise with binary cross-entropy; ground-truth contacts are defined by thresholding measured forces at 2N. When used, the joint-state decoder predicts $(\mathbf{q})$ (14D with sin/cos representation for $\mathbf{q}$) with an ℓ_2 loss. We train end-to-end (encoders + predictor + decoders) using AdamW with learning rate $5e-4$ (predictor), batch size $B = 32$, and 100 epochs. Segments are sampled as $(L + H)$-length windows with random offsets.

A. Two-fidelity rollouts for real-time control

a) Draft model design and training.: The draft predictor $\mathbf{F}_l^{\mathrm{draft}}$ shares the same encoders and action embedding ϕ_{act} as the full predictor, but uses a lightweight two-layer MLP. We train $\mathbf{F}_l^{\mathrm{draft}}$ purely using teacher forcing loss (imitate the full model).

b) Selection and rescoring.: At each MPC step t, we sample N action sequences $\{\mathbf{u}_{t:t+H-1}^{(j)}\}_{j=1}^N$. We evaluate the analytical rollout $\mathbf{F}_k$ for all N particles. For the learned rollout, we: (i) roll out $\mathbf{F}_l^{\mathrm{draft}}$ for all particles to obtain draft predictions and costs $\{\mathcal{J}_j^{\mathrm{draft}}\}_{j=1}^N$, (ii) select the top-$M$ candidates $\mathcal{I}_M = \mathtt{TopM}(\{\mathcal{J}_j^{\mathrm{draft}}\}, M)$ with $M = 10$ and $\mathcal{I}_M \subset \{1, \ldots, N\}$, and (iii) re-roll out only those candidates with the full predictor $\mathbf{F}_l$ over the full horizon to compute $\{\mathcal{J}_j^{\text{full}}\}_{j\in\mathcal{I}_M}$. We form the final costs used by MPPI as

$$\mathcal{J}_j = \begin{cases} \mathcal{J}_j^{\text{full}}, & j \in \mathcal{I}_M, \\ \mathcal{J}_j^{\text{draft}}, & \text{otherwise.} \end{cases}$$

c) Draft-error safeguard and fallback policy.: To detect when the draft model is unreliable, we monitor a one-step latent prediction error at runtime:

$$e_t \ = \ \|\hat{\mathbf{z}}_{t+1}^{\text{draft}} - \mathbf{z}_{t+1}\|_2,$$

where $\mathbf{z}_{t+1} = \phi(o_{t+1}^{\text{cc}})$ is computed from the next observation. If $e_t > \tau_{\text{draft}}$ with $\tau_{\text{draft}} = 0.1$, we disable the two-fidelity scheme and evaluate the full model for all particles (equivalently, set $M = N$) for the next $T_{\text{full}} = 4$ MPC steps.

E. Rollout Cost Details

A. Kinematic cost c_{kin}

We decompose $c_{\text{kin}}(\hat{s}_{t+i})$ into feasibility and task terms:

$$c_{\text{kin}}(\hat{s}_{t+i}) = \lambda_{\text{lim}}\, c_{\text{lim}}(\hat{\mathbf{q}}_{t+i}) + \lambda_{\text{coll}}\, c_{\text{coll}}(\hat{\mathbf{q}}_{t+i}) + \lambda_{\text{ee}}\, c_{\text{ee}}(\hat{\mathbf{e}}_{t+i}) + \lambda_u\, c_u(\mathbf{u}_{t+i}), \tag{1}$$

with $\lambda_{\text{lim}} = 1000.0$, $\lambda_{\text{coll}} = 1000.0$, $\lambda_{\text{ee}} = [1.5.0, 10.0]$ (orientation, position), and $\lambda_u = 1.0$.

a) Joint limits.: Let $(\mathbf{q}_{\min}, \mathbf{q}_{\max})$ be joint bounds and Δ_q a soft margin. We use

$$c_{\text{lim}}(\hat{\mathbf{q}}) = \sum_{m=1}^{n} \Big([\hat{q}_m - (q_{\max,m} - \Delta_q)]_+^2 + [(q_{\min,m} + \Delta_q) - \hat{q}_m]_+^2 \Big), \tag{2}$$

with $\Delta_q = 0.05\,(q_{\max,m} - q_{\min,m})/2$.

b) Self-collision.: We follow STORM's implementation and learn a lightweight collision classifier for our robot model.

c) Task-space tracking (if specified).: When a task-space end-effector target pose $\mathbf{e}_g = (\mathbf{p}_g, \mathbf{R}_g)$ is defined, we penalize translation and rotation:

$$c_{\text{ee}}(\hat{\mathbf{e}}) = \frac{\|\hat{\mathbf{p}} - \mathbf{p}_g\|_2^2}{\sigma_p^2} + \frac{\|\mathbf{I} - \mathbf{R}_g^\top \hat{\mathbf{R}}\|_F^2}{\sigma_R^2}. \tag{3}$$

B. Active force cost

Let $\mathcal{K}_t$ denote the active contact set at time t. For each $k \in \mathcal{K}_t$, we form a spring-based force estimate along the measured contact normal using the kinematic rollout $\hat{\mathbf{q}}_{t+1:t+H}$. For taxel k, let $\mathbf{J}_{k,t}^n = \mathbf{n}_{k,t}^\top \mathbf{J}_{k,t}$ be the normal Jacobian row. Given a kinematic rollout $\hat{\mathbf{q}}_{t:t+H}$, define $\Delta\mathbf{q}_{t+i} = \hat{\mathbf{q}}_{t+i} - \hat{\mathbf{q}}_{t+i-1}$ and the predicted normal displacement

$$\Delta x_{k,t+i}^n = \mathbf{J}_{k,t+i}^n\, \Delta\mathbf{q}_{t+i}. \tag{4}$$

We treat motion into contact as compression and accumulate

$$\hat{\delta}_{k,t+i} = \max\big(0,\ \hat{\delta}_{k,t+i-1} - \Delta x_{k,t+i}^n\big), \qquad \hat{\delta}_{k,t} = 0, \tag{5}$$

then estimate normal force with a linear spring

$$\hat{F}_{k,t+i} = k_{\text{spr}}\, \hat{\delta}_{k,t+i}, \tag{6}$$

where k_{spr} is fit offline. For active contacts, we initialize the spring compression using the measured force at time t to reduce bias in the rollout. For each $k \in \mathcal{K}_t$ with measured normal force $f_{k,t}$, we set

$$\hat{\delta}_{k,t} = \text{clip}\bigg(\frac{f_{k,t}}{k_{\text{spr}}},\ 0,\ \delta_{\max}\bigg), \tag{7}$$

and then propagate $\hat{\delta}_{k,t+i}$ using Eq. (5). The resulting force estimate is $\hat{F}_{k,t+i} = k_{\text{spr}}\hat{\delta}_{k,t+i}$. We penalize violations of the safety threshold F_{safe} over the active contacts with

$$c_{\text{active}}(\hat{\mathbf{q}}_{t+i}) = \frac{1}{|\mathcal{K}_t|} \sum_{k\in\mathcal{K}_t} \big[\hat{F}_{k,t+i} - F_{\text{safe}}\big]_+^2. \tag{8}$$

C. Predicted force safety cost c_{force}

Raw force predictions from the learned decoder can be noisy when evaluated far into the horizon. To obtain a more stable force signal for scoring, we first use the binary contact predictor to estimate the contact onset time $\hat{\tau}_k$ for each taxel k. We record the predicted joint configuration at onset, $\hat{\mathbf{q}}_{t+\hat{\tau}_k}$, and treat it as the reference configuration for subsequent force estimation. We then apply the same spring model (with stiffness k_{spr} as above) to convert predicted compression after onset into a per-taxel normal force estimate $\hat{\mathbf{f}}_{t+i} \in \mathbb{R}^{N_s}$.

We penalize predicted forces above the safety threshold F_{safe} with a hinge-squared penalty:

$$c_{\text{force}}(\hat{\mathbf{f}}_{t+i}) = \frac{1}{N_s} \sum_{k=1}^{N_s} \big[\hat{\mathbf{f}}_{k,t+i} - F_{\text{safe}}\big]_+^2. \tag{9}$$

D. Latent goal cost c_{goal}

We encode the goal observation as $\mathbf{z}_g = \phi(o_g^{\text{cc}})$, where o_g^{cc} is formed analogously to o_t^{cc} (Sec. B). The goal cost is

$$c_{\text{goal}}(\hat{\mathbf{z}}_{t+i}, \mathbf{z}_g) = \|\hat{\mathbf{z}}_{t+i} - \mathbf{z}_g\|_2^2. \tag{10}$$

E. Weights and implementation

Costs are summed over the horizon with discount factor $\gamma = 0.98$. Infeasible rollouts (large joint-limit violations or self-collision) are handled by increasing the corresponding penalty by a factor of 1000 rather than rejecting the sample.

F. IQL Value Learning for Long-Horizon Tasks

A. Progress Variables and Subgoals

For each long-horizon task, the dataset provides low-dimensional progress variables $\xi_t \in \mathbb{R}^{d_g}$ (e.g., body turn angle, limb joint angles). We obtain the values for these indicators during data collection by mounting ArUco markers on the head and limb joints. Note that these markers and not required at test time. Instead, we train progress heads that predict these progress variables for current $\mathbf{z}_t$. At test time we specify an ordered subgoal sequence $\{\xi_g^{(1)}, \dots, \xi_g^{(G)}\}$ and maintain an active index g during execution. We advance to the next subgoal when the measured progress satisfies

$$\|\xi_t - \xi_g^{(g)}\|_2 \le \delta_\xi, \tag{11}$$

for a fixed threshold δ_ξ (task-dependent).

B. Latent Encoding and Goal Representation

We freeze the multimodal encoders (Sec. B) and compute a latent embedding $\mathbf{z}_t$ for each observation. For each subgoal $\xi_g^{(g)}$, we also form a corresponding goal observation and embed it with the same frozen encoders to obtain the goal latent $\mathbf{z}_g$. The choice of subgoals depends on task difficulty. For instance, in case of rollover, the subgoals are chosen as the two intermediate frames where contact switching happens (robot pulls from near the knee and then moves closer to the lower back and pulls from there).

C. Offline Transition Dataset

From the expert dataset, we construct an offline transition set

$$\mathcal{D} = \{(\mathbf{z}_t, \mathbf{u}_t, \mathbf{z}_{t+1}, \mathbf{z}_g, r_t, d_t)\}, \tag{12}$$

where $\mathbf{u}_t$ is the executed action, $d_t \in \{0, 1\}$ indicates episode termination, and the reward is defined as the negative of the cost used for scoring task state (Sec. **??**):

$$r_t \;=\; -\ell_t. \tag{13}$$

ℓ_t includes a progress/subgoal tracking term built from the labeled ξ_t and current subgoal $\xi_g^{(g)}$, e.g.,

$$\ell_{\text{prog}}(\xi_t, \xi_g^{(g)}) = \|\xi_t - \xi_g^{(g)}\|_2^2, \tag{14}$$

For rollover, this is the L2 error between the goal head angle and current head angle (predicted by the progress head). In case of limb repositioning, this is the L2 error between goal limb configuration and current limb configuration.

D. Goal-Conditioned IQL Critic

We implement goal-conditioning by concatenating the state and goal latents (and optionally their difference):

$$x_t = [\mathbf{z}_t,\ \mathbf{z}_g,\ (\mathbf{z}_g - \mathbf{z}_t)]. \tag{15}$$

We learn double action-value functions $Q_{\theta_1}(x_t, \mathbf{u}_t)$, $Q_{\theta_2}(x_t, \mathbf{u}_t)$ and a value function $V_\phi(x_t)$, with a slowly-updated target value network $V_{\bar{\phi}}$.

a) Q-learning objective.: The TD target is

$$y_t = r_t + \gamma(1 - d_t)\, V_{\bar{\phi}}(x_{t+1}), \tag{16}$$

and we minimize

$$\mathcal{L}_Q = \mathbb{E}_{\mathcal{D}}\Big[\big(Q_{\theta_1}(x_t, \mathbf{u}_t) - y_t\big)^2 + \big(Q_{\theta_2}(x_t, \mathbf{u}_t) - y_t\big)^2\Big]. \tag{17}$$

b) Expectile value objective.: Following IQL, V_ϕ is trained via expectile regression toward the in-dataset action values:

$$\mathcal{L}_V = \mathbb{E}_{\mathcal{D}}[\rho_\tau(\min(Q_{\theta_1}, Q_{\theta_2})(x_t, \mathbf{u}_t) - V_\phi(x_t))], \tag{18}$$

where ρ_τ is the expectile loss with expectile parameter $\tau \in (0, 1)$.

We use discount $\gamma = 0.99$, expectile $\tau = 0.7$, and Polyak target update rate $\alpha = 0.005$. All networks ($Q_{\theta_1}, Q_{\theta_2}, V_\phi$, and the AWR actor π_ψ) are 2-layer MLPs, dropout $p = 0.1$, and ReLU activations. We train all networks with AdamW (lr $= 3{\times}10^{-4}$). The advantage-weighted regression temperature is $\beta_{\text{AWR}} = 3.0$.

E. Using the Value for Terminal Shaping in MPPI

At test time, MPPI rolls out the learned latent predictor $\mathbf{F}_l$ to obtain the horizon-end latent $\hat{\mathbf{z}}_{t+H}$ for each sampled action sequence. We then augment the rollout cost as

$$\mathcal{J}_{\text{aug}}(\mathbf{u}_{t:t+H-1}) = \mathcal{J}(\mathbf{u}_{t:t+H-1}) - \lambda_V\, V_\theta(\hat{\mathbf{z}}_{t+H}, \mathbf{z}_g), \tag{19}$$

where V_θ is the learned IQL value function. Since V_θ estimates reward-to-go and MPPI minimizes cost, the negative sign converts reward-to-go into a cost-to-go term. We set $\lambda_V = 0.1$.

Method	**SR**
TACTIC	$6/10$
TACTIC (w/o Value Fn.)	$2/10$

TABLE IV: Ablation on the effect of the terminal value function, reported as success rate (SR) over 10 trials.

Ablation with value function. We ablate the terminal value function in TACTIC to assess its contribution to task success, reporting success rate (SR) over 10 trials.

G. Tactile Skin: Hardware, Frames, Pose/Normal Calibration, and Limitations

A. Hardware overview

We mount 22 Tekscan FlexiForce A502 force-sensing resistors (FSRs) on the robot to measure distributed normal contact forces along the arm. Each pad measures normal force over a $50.8\,\text{mm} \times 50.8\,\text{mm}$ sensing area and outputs a resistance change proportional to applied load. Sensors are grouped into four physical mounts: the two wrist joints (5 pads each), the forearm link (6 pads), and the upper-arm link (6 pads). Each mount is instrumented with an Adafruit Feather M0 WiFi board that streams raw sensor readings over UDP at 800 Hz (per board).

B. Sensor readings and preprocessing

Taring. Before interaction, we tare each sensor by estimating its no-contact offset from a recent window of raw readings. A tare is accepted only when the window variance is below a stability threshold.

Filtering. We apply a median filter followed by a low-pass filter to the tared signal. The resulting tactile vector $\mathbf{f}_t \in \mathbb{R}^{N_s}$ is published at 100 Hz for control and logging.

C. Taxel pose, normals, and contact Jacobians

Taxel extrinsics. For each taxel k, we define a taxel frame $\{T_k\}$ attached to the sensing pad and store its fixed transform with respect to the corresponding link frame $\{L\}$, ${}^{L}\mathbf{T}_{T_k}$. We obtain taxel positions and outward normals in the link frame from manual measurements (vernier caliper) and CAD alignment, and include these extrinsics in the robot model.

Runtime calculation. At runtime, forward kinematics provides the link pose ${}^{W}\mathbf{T}_L(\mathbf{q}_t)$, which we use to transform each taxel pose and normal into the world frame and to compute the translational Jacobian at the taxel center. We validate these transforms by visualizing taxel frames across diverse arm configurations.

D. Force calibration

We calibrate the skin sensor using an ATI Nano25 force/torque sensor. We repeatedly apply normal loads over a range of magnitudes and fit a linear model that maps raw readings to force in Newtons using least-squares regression..

E. Driver and data integrity checks

We implement a `skin_driver` that receives UDP packets from the WiFi boards and exposes ROS services and topics for taring and reading tactile values. The driver includes a watchdog for common failure modes (e.g., missing packets, low battery, disconnected boards, inactive channels). The driver performs the filtering discussed above and publishes the filtered forces as an array.

F. Limitations

Coverage. The tactile skin provides discrete coverage over a subset of the arm surface; contact can occur in uninstrumented regions. **Normal-only sensing.** The sensors measure normal force only. This ignores tangential forces and frictional effects (e.g., shear during sliding), which can be informative for both scoring and sampling in contact-rich tasks. **System dependence.** Our method benefits from improved tactile coverage, higher bandwidth, and better calibration. The core algorithm does not assume a specific sensor type, but performance will improve depending on the quality of contact localization and force estimation available at runtime.

H. Data Collection Details

A. Exoskeleton teleoperation interface

We collect human demonstration data using a custom exoskeleton teleoperation interface that enables mirrored whole-arm control of a Kinova Gen3 (7-DoF). Mirroring human arm motion is important for capturing whole-arm contact strategies relevant to caregiving, including distributing contact across links and coordinating intermediate-link and end-effector motion. The exoskeleton is a scaled replica of the robot and matches its kinematic structure using seven Dynamixel XC330-M288-T actuators. An additional Dynamixel XL330-M077-T actuator in a trigger handle is used to start and stop data collection. We include safety checks and a paired initialization procedure to ensure consistent alignment between the exoskeleton and robot before teleoperation.

B. Offline dataset summary

During data acquisition, RGB-D, tactile, and proprioceptive observations are recorded synchronously at 40 Hz. RGB and depth images are captured with an Intel RealSense D455 at 60 Hz and time-synchronized to the 40 Hz logging stream. In the 3D Maze environment, we collected 125 expert and 125 random-play episodes (each $\sim$30 s). For Side Rollover, we collected 100 expert and 100 random-play episodes. For Limb Repositioning, we collected 100 expert and 100 random-play episodes. Episodes in the bed-based environments have an average duration of $\sim$45 s. Expert trajectories correspond to successful task completion (e.g., reaching the specified goals in the maze), while random-play trajectories consist of exploratory motions that still progress toward task-relevant goal states.

I. Experiment Details

A. V-JEPA2 and DINO-WM experiment details

Finetuning V-JEPA2 and Hybrid Model Experiment. We load the pre-trained encoders and perform action-conditioned finetuning of the predictor using the tabletop reach dataset. To make the model hybrid, we do FK to compute end-effector samples and add a cost for end-effector goal reaching. This cost forces the planner to blindly move along the line connecting the current state and goal state, thus not reasoning about collision avoidance at all. We combine this with the latent goal reaching cost which intuitively encodes the avoidance behavior seen in the dataset. Upon tuning the weights for 5 different start-goal configurations (not included in testing) we fix the cost weights and run evaluations with models that are trained on different %ages of finetuning data. We also run the experiment with different obstacle shapes and sizes, and obtain similar performance improvements since the kinematics-based policy is independent of these effects in the data.
DINO-WM Granular Experiment. For this experiment, we train a policy using the codebase and dataset provided by the authors of DINO-WM. To make the model hybrid, we add an end-effector pose cost derived from a simple kinematic heuristic. At each step, the heuristic compares the current and goal particle-density maps, identifies the top-K surplus and deficit regions, selects the surplus-deficit pair with the shortest unobstructed path (estimated via HSV-based masking and downsampled frame), and generates a greedy push from surplus to deficit. The resulting push direction and target pose define the additional end-effector cost term.