

Beyond the End-Effector: Learning Whole-Arm Contact Manipulation from Human Demonstrations

Bo-Ruei Huang, Daniel Seita, Erdem Biyik

Thomas Lord Department of Computer Science, University of Southern California
{borueihu, seita, biyik}@usc.edu

Abstract: Robot manipulation has traditionally relied on end-effector-centric interactions. In contrast, humans routinely exploit whole-arm contacts to improve efficiency, robustness, and adaptability, especially when the hands are occupied or when precise grasping is unnecessary. Enabling similar capabilities in robots remains challenging: the robot must select where on its arm to make contact, generate effective non-prehensile interactions, and satisfy task-specific pose or safety constraints on the end-effector. We propose *Whole-Arm Contact Learning* (WACL), a hierarchical framework that describes each whole-arm contact task with three *robot-agnostic* representations and hands them to a quasi-static surface-contact optimizer: semantic keypoints that summarize the scene, a reference contact trajectory that specifies where and how the arm presses on the object, and safety-aware path constraints that a vision-language model generates from the task description to protect the end-effector’s task. The robot’s kinematic chain enters only in this optimizer, which grounds these representations onto the arm by jointly searching over which link makes contact, where on that link, and the joint configuration that realizes it. This decomposition lets WACL learn the contact trajectory from 10 *bare-hand human* demonstrations per task collected without teleoperation, while new end-effector constraints change only the optimizer’s input rather than any trained component. Real-world experiments on a 7-DoF arm across four whole-arm tasks show that WACL sustains contact under end-effector constraints where a strong link-pose baseline fails (78% vs. 33% success) and generalizes to new constraints without retraining. Website: <https://wac1-robot.github.io>.

Keywords: Robot manipulation, Learning and planning in robotics, Whole-arm manipulation, Learning from demonstrations

1 Introduction

In our daily lives, manipulation is rarely confined to the fingertips. When our hands are already occupied, such as when carrying a load or holding a cup of coffee, we routinely recruit the rest of the arm to act on the environment, nudging a door open with an elbow or steadying a box against a forearm. This “Whole-Arm Manipulation” (WAM) [1] lets us take on a second task without abandoning the first: the hand keeps the cup upright—the *primary* task—while the arm presses a button or pushes a drawer shut—the *secondary* task. This multi-task setting introduces a constraint: whatever the arm does, it must not disturb what the hand is already committed to.

Robot manipulation, however, remains overwhelmingly end-effector-centric. Pre-trained vision-language-action models (VLAs) typically output gripper-pose actions [2, 3, 4, 5, 6, 7, 8] and treat the arm as a mechanism for transport, so arm-environment contact is usually avoided, let alone utilized. This discards useful kinematic redundancy and limits the robot in constrained or cluttered settings, where whole-arm contact may be necessary because a gripper-only motion is infeasible. Learning whole-arm manipulation end-to-end is also difficult for two compounding reasons. First, kinematically similar contacts on different links (e.g., elbow, forearm, upper arm) produce substantially different pixel observations and joint configurations, so transferring contact behavior across links is difficult

for a policy trained on raw inputs, a within-robot generalization gap. Second, while people regularly engage in whole-arm contact, teleoperating a 7-DoF arm into the same contacts is slow and unintuitive, and typically requires a leader-follower system [9] or an exoskeleton [10]. Bare-hand human demonstrations are therefore a more natural data source, but they provide no robot actions [11, 12, 13, 14, 15] and introduce a human-to-robot embodiment gap.

We propose *Whole-Arm Contact Learning* (WACL), a hierarchical framework that closes both gaps by describing each task with three robot-agnostic representations and handing them to a surface-contact trajectory optimizer: *semantic keypoints* that summarize the scene, a *reference contact trajectory* that specifies where and how the arm should press on the object, and *safety-aware path constraints* on the end-effector that keep its own task valid. The robot’s kinematic chain enters only once, in this optimizer, which grounds these representations onto the arm by jointly searching over which link makes contact, where on that link, and the joint configuration that realizes it, as illustrated in Figure 1. This decomposition lets WACL learn contact trajectories from bare-hand human demonstrations, while new end-effector constraints change only the optimizer’s input, not the trained policy.

We evaluate WACL on four real-world whole-arm tasks on a 7-DoF xArm, each requiring a secondary contact action while preserving a primary end-effector task (e.g., turning a faucet off while keeping a wet pad above the sink). Without end-effector constraints, WACL performs on par with or above a strong keypoint-conditioned baseline, KALM [16] adapted to regress the contact link’s pose; under constraints, WACL more than doubles the baseline’s success rate (78% vs. 33%) by re-routing the contact to a link and anchor that keep the primary task intact. WACL further handles a second, simultaneous constraint with no retraining and transfers zero-shot to a second arm, a Franka FR3, with only the optimizer’s robot model changed; an ablation shows that the sliding surface anchor accounts for part of its constrained performance.

Our contributions are summarized as follows:

- WACL, a hierarchical framework that decomposes whole-arm manipulation into three robot-agnostic representations: semantic keypoints, a reference contact trajectory, and safety-aware path constraints, so that the robot’s kinematic chain is grounded in a single downstream optimizer.
- A robot-agnostic data pipeline that learns the contact trajectory from bare-hand, multi-view human demonstrations, with no whole-arm teleoperation.
- A surface-contact trajectory optimizer that jointly searches over the contact link, a sliding surface anchor, and the joint configuration under path constraints generated by a vision-language model (VLM), with real-world experiments showing sustained contact under end-effector constraints.

2 Related work

Whole-body manipulation and sensing. Whole-body control for humanoids, quadrupeds, and mobile manipulators coordinates many degrees of freedom [17, 18, 19, 20], but its focus is balance, locomotion, and floating-base coordination rather than exploiting limb surfaces for functional interaction. Contact-rich manipulation work explores tactile feedback [21] and compliant skins [22, 23, 24] for richer body-environment interaction, but these assume specialized hardware. Fewer works treat the arm’s surface (e.g., forearm, elbow) as the manipulation interface. Learning-based methods collect joint-space demonstrations through an exoskeleton kinematically matched to the robot’s arm [10, 25] or hand [26], or train per-task sim-to-real reinforcement-learning policies for a compliant, tactile-skinned humanoid [27]. Model-based methods plan contacts from known object models by global search over smoothed quasi-dynamic contact models [28], tensor-factorized tree search [29], or continuous optimization over an explicit robot-surface representation [30]. Such a planner has also been distilled into a diffusion policy trained in simulation [31]. To our knowledge, none holds a concurrent end-effector task while the arm makes contact. Our framework instead learns the contact trajectory from bare-hand human demonstrations, needs no robot demonstrations, object model, or tactile sensing, and enforces the end-effector task inside the optimizer that chooses the contact link.

Vision-language models (VLMs) for robotics. Foundation models, including VLMs, have proven useful for robotic reasoning, enabling systems to perform spatial planning [32, 33], synthesize control code [34], craft reward functions [35, 36, 37], and specify geometric constraints [38, 39,

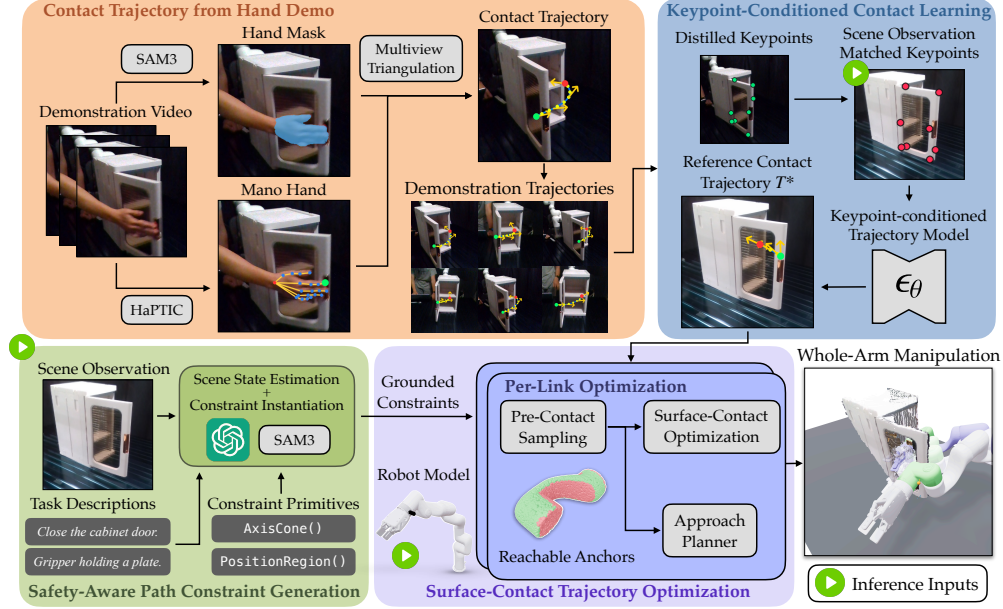


Figure 1: **WACL overview.** WACL produces three robot-agnostic representations and grounds them onto the robot in a single downstream optimizer. **Contact trajectory from hand demonstrations.** A segmentation model and 4D hand tracker fuse bare-hand multi-view RGB-D demonstrations into a *reference contact trajectory* of contact positions and surface normals. **Keypoint-conditioned contact learning.** Semantic keypoints distilled by KALM are matched to the scene, conditioning a trajectory model that generates the contact trajectory. **Safety-aware path constraint generation.** A VLM grounds a fixed vocabulary of primitives into path constraints that preserve the primary end-effector task. **Surface-contact trajectory optimization.** The optimizer searches each link’s reachable anchors with surface-contact inverse kinematics and beam search under the constraints.

40]. Recent work also moves beyond collision-free reasoning to use VLMs for contact semantics, inferring which parts of an environment may tolerate interaction [41]. These methods constrain or plan the end-effector; none selects a contact link on the arm or protects a task the end-effector is already performing. Hierarchical frameworks ground high-level semantic plans in learned low-level policies [16, 42, 43, 44]. We apply VLM-generated constraints at the *link level*: they must hold on one link while a different link makes contact, and changing them never retrains the policy.

3 Method

We propose *Whole-Arm Contact Learning* (WACL), a hierarchical framework that decomposes whole-arm manipulation into three stages. Section 3.1 defines the task representations and the constrained optimization they induce. WACL learns the reference contact trajectory from bare-hand demonstrations (Section 3.2), instantiates path constraints from the task description and scene observation (Section 3.3), and optimizes a feasible whole-arm trajectory (Section 3.4). Section 3.5 combines the three stages into the end-to-end inference pipeline. See Figure 1 for an overview.

3.1 Problem formulation

WACL represents each task with three *robot-agnostic* representations, deliberately free of any particular robot’s kinematics. The *semantic keypoints* $K_{\text{obs}} = \{\mathbf{p}_i\}_{i=1}^{n_{\text{kp}}} \in \mathbb{R}^{n_{\text{kp}} \times 3}$ summarize the observation as 3D scene points that localize task-relevant geometry independent of the robot’s pose (Section 3.2). The *reference contact trajectory* $\mathcal{T}^* = \{(\mathbf{p}_t^*, \mathbf{n}_t^*)\}_{t=1}^H$ specifies a world-frame contact position and surface normal at each of H waypoints, with no commitment to which link makes contact or to any joint configuration (Section 3.2). The *safety-aware path constraints* $\{\mathcal{C}_k\}$ are typed requirements on link-level quantities (e.g., “the gripper axis stays within 25° of world up”), evaluated at every waypoint (Section 3.3).

The optimizer acts on a robot with configuration space $\mathcal{Q} \subset \mathbb{R}^{n_q}$ and joint configuration $q_t \in \mathcal{Q}$ over a horizon of H waypoints covering the contact segment. To let it decide *which* link makes contact, we expose the contact-capable links explicitly: each link $\ell \in \mathcal{L}$ carries a collision mesh $\mathcal{M}_\ell$, and

their disjoint union $\mathcal{M} = \bigsqcup_{\ell \in \mathcal{L}} \mathcal{M}_\ell$ is the robot’s contact surface, with every point tagged by the link it belongs to. A *surface anchor* is a point $\mathbf{u} \in \mathcal{M}_\ell$ in ℓ ’s local frame, whose world-frame position and normal at any q follow from forward kinematics (Section 3.4).

Objective. At inference the system receives an observation $O = (I, d_{\text{task}})$, a calibrated RGB-D image and a natural-language task description, from which it produces $(K_{\text{obs}}, \mathcal{T}^*, \{\mathcal{C}_k\})$. Given $\mathcal{T}^*$ and $\{\mathcal{C}_k\}$, we seek a joint trajectory $q_{1:H}$, a contact link $\ell \in \mathcal{L}$, and a per-waypoint anchor sequence $\mathbf{u}_{1:H}$ such that the world-frame contact at each waypoint tracks $(\mathbf{p}_t^*, \mathbf{n}_t^*)$, every $\mathcal{C}_k$ is satisfied, and the trajectory is kinematically and collision feasible. The free-space approach from the robot’s home pose to the start of the contact segment is planned separately (Section 3.4).

3.2 Contact trajectory learning

Because $\mathcal{T}^*$ names only a contact position and normal, a bare hand under multi-camera RGB-D capture can produce it directly. This subsection describes the offline pipeline that turns such demonstrations into the $(K_{\text{obs}}, \mathcal{T}^*)$ pairs used to train the keypoint-conditioned policy π_θ .

3.2.1 Offline data pipeline

Multi-view hand demonstration capture. A human performs each task bare-hand under C calibrated RGB-D cameras. Each demonstration is automatically segmented to its contact phase by a fingertip-velocity threshold; the operator can mark the contact boundary by pausing without extra cost. The human contacts the object with a fingertip, a point proxy for the robot’s link-surface contact.

Per-view hand pose lifting. For each per-camera RGB-D stream, we segment the hand with SAM3 [45] and regress MANO [46] parameters using the 4D hand tracker HaPTIC [47]. The tracks are temporally coherent, but each view is monocular and therefore inaccurate in depth [11], while the contact trajectory needs accurate 3D fingertip positions.

Multi-view hand fusion. To address the problem, we fuse the per-view estimates by triangulating the contact fingertip vertex across the C calibrated views via per-point weighted-midpoint ray triangulation [48], with each camera weighted by the silhouette intersection-over-union (IoU) between the reprojected MANO mesh and the SAM3 hand mask in that view. Because each camera’s depth uncertainty lies along its viewing ray, triangulation drives the residual into directions the other cameras constrain laterally, yielding a single world-frame fingertip trajectory consistent across views.

Reference contact trajectory extraction. We resample the fused fingertip trajectory to the fixed horizon H via cubic spline, and pair each sampled position $\mathbf{p}_t^*$ with the corresponding MANO surface normal $\mathbf{n}_t^*$ at the contact fingertip to obtain the reference contact trajectory $\mathcal{T}^* = \{(\mathbf{p}_t^*, \mathbf{n}_t^*)\}_{t=1}^H$ for the demonstration. The resulting $\mathcal{T}^*$ does not refer to the robot, so the same demonstrations serve any arm; Appendix A.1 details capture, fusion, and extraction.

3.2.2 Keypoint distillation and policy training

Semantic keypoint distillation. The observation representation K_{obs} is produced by the KALM keypoint distillation pipeline [16]. One demonstration is selected as the reference and its task-relevant region is segmented by a VLM-driven SAM3 prompt; in every other demonstration, DINOv2 [49] features and FPFH [50] geometric consistency localize a fixed set of n_{kp} keypoints corresponding to the reference. At inference, the same matcher Φ_{kp} is applied to a live RGB-D view and recovers each keypoint by maximizing feature similarity against the reference,

$$\mathbf{p}_i = \arg \max_{\mathbf{p}} \langle F_{\text{obs}}(\mathbf{p}), F_{\text{ref}}(\mathbf{k}_i) \rangle, \quad K_{\text{obs}} = \{\mathbf{p}_i\}_{i=1}^{n_{\text{kp}}}, \quad (1)$$

where F_{obs} and F_{ref} are the combined DINOv2 and FPFH features of the query and reference views and $\{\mathbf{k}_i\}_{i=1}^{n_{\text{kp}}}$ are the reference keypoints.

Keypoint-conditioned policy. We adopt KALM [16] as a conditional trajectory diffusion policy π_θ that models $p(\mathcal{T}^* | K_{\text{obs}})$, trained with a noise-prediction objective in the style of Diffuser [51] on the $(K_{\text{obs}}, \mathcal{T}^*)$ pairs from Section 3.2.1. At inference, the policy iteratively denoises Gaussian noise into a reference contact trajectory, $\mathcal{T}^* \sim \pi_\theta(\cdot | K_{\text{obs}})$.

3.3 Safety-aware path constraint generation

The reference contact trajectory $\mathcal{T}^*$ specifies *where* on the scene a chosen link must make contact, but says nothing about the rest of the arm. Whole-arm manipulation typically arises when the end-effector is already committed to a primary task, such as carrying a cup, and the contact action must not disturb it. The path constraints $\{\mathcal{C}_k\}$ encode this remaining requirement: they restrict the pose of links *other than* the contact link throughout the contact segment so that the primary end-effector task is preserved regardless of what the contact link is doing. We populate $\{\mathcal{C}_k\}$ at inference from a fixed, typed constraint vocabulary whose parameters are filled from an open-vocabulary scene state via two GPT-5.5 calls, one for scene state estimation and one for constraint instantiation.

Constraint primitives. Each constraint $\mathcal{C}_k$ instantiates a primitive from a small fixed vocabulary, a requirement on one link’s axis or position, with parameters ϕ_k . Its violation $c_k(q; \phi_k) \geq 0$ (written $c_k(q)$ below) is zero when the requirement holds at q and grows with the amount by which it is missed. The optimizer uses this one quantity twice: added to the objective as a soft cost, so partial violations have a gradient to descend, and as the hard test $c_k(q_t) \leq \tau_k$ at every waypoint, so no weighting of the contact objective can push a violation past the tolerance τ_k . Primitive types per task are listed in Section 4.1; deployed parameters and tolerances in Appendix A.3.

Open-vocabulary scene state estimation. The first VLM call takes (I, d_{task}) and enumerates the task-relevant objects in the scene via chain-of-thought decomposition [52]. Each named object is segmented by SAM3 [45] into a 2D mask, back-projected into a 3D point cloud using the depth channel of I , and reduced to an oriented bounding box. The result is a scene state $\mathcal{S}$, a list of named entities with oriented bounding boxes that fill the primitives’ geometric parameters.

Constraint instantiation. The second VLM call receives d_{task} , the scene state $\mathcal{S}$, and the world-frame end-effector pose from forward kinematics. It selects a primitive for each requirement in d_{task} , fills ϕ_k from $\mathcal{S}$, and returns a typed list parsed into $\{\mathcal{C}_k\}$; Appendix A.4 gives the prompts.

3.4 Surface-contact trajectory optimization

This subsection instantiates the constrained optimization of Section 3.1: jointly determine a contact link ℓ , an anchor sequence $\mathbf{u}_{1:H}$, and a joint trajectory $q_{1:H}$ so that the resulting world-frame contact tracks $\mathcal{T}^*$ at every waypoint while satisfying every $\mathcal{C}_k$. The three decision variables are qualitatively different — ℓ is discrete, each $\mathbf{u}$ is confined to a 2D mesh surface, and each q is continuous — so a gradient-only formulation is poorly suited to the joint search. We instead adopt a sample-and-search formulation: discretize the per-link contact mesh into a candidate anchor set, solve a surface-contact inverse kinematics problem at each anchor and waypoint to obtain admissible joint configurations, and assemble these into a trajectory via beam search. Our first attempt, a continuous program over the joint trajectory and a contact point blended across mesh triangles, rarely converged, as Appendix A.5 details. Sampling anchors and solving per-waypoint inverse kinematics (IK) in parallel on the GPU prunes infeasible branches early, at the cost of a greedy search smoothed only by the step cost of Equation (5). For link ℓ at joint configuration q , forward kinematics returns the $SE(3)$ transform

$$T_\ell(q) = \begin{bmatrix} R_\ell(q) & t_\ell(q) \\ \mathbf{0}^\top & 1 \end{bmatrix}, \quad \mathbf{p}(q, \mathbf{u}) = T_\ell(q) \mathbf{u}, \quad \mathbf{n}(q, \mathbf{u}) = R_\ell(q) \mathbf{n}^{\text{local}}(\mathbf{u}), \quad (2)$$

where $\mathbf{n}^{\text{local}}(\mathbf{u})$ is the outward mesh normal at the link-local anchor $\mathbf{u}$, and the full objective is

$$\min_{q_{1:H}, \ell, \mathbf{u}_{1:H}} \sum_{t=1}^H J_{\text{contact}}(q_t, \mathbf{u}_t; \mathbf{p}_t^*, \mathbf{n}_t^*) \quad \text{s.t.} \quad c_k(q_t) \leq \tau_k \quad \forall k, t; \quad q_t \in \mathcal{Q}; \quad \text{collision-free}, \quad (3)$$

with J_{contact} the per-waypoint surface-contact cost defined below.

Reachable anchor set per link. For each contact link ℓ , we precompute a candidate anchor set on $\mathcal{M}_\ell$ by farthest-point sampling [53] its collision mesh, the same geometry the planner uses for self-collision checks so that every candidate is physically meaningful. We then prune anchors that are unreachable in practice: those obstructed by adjacent links along their outward normal, those lying inside the signed-distance volume of a neighboring link, and those recessed inside the convex hull of $\mathcal{M}_\ell$. The remaining anchors form the discrete search space the optimizer expands over.

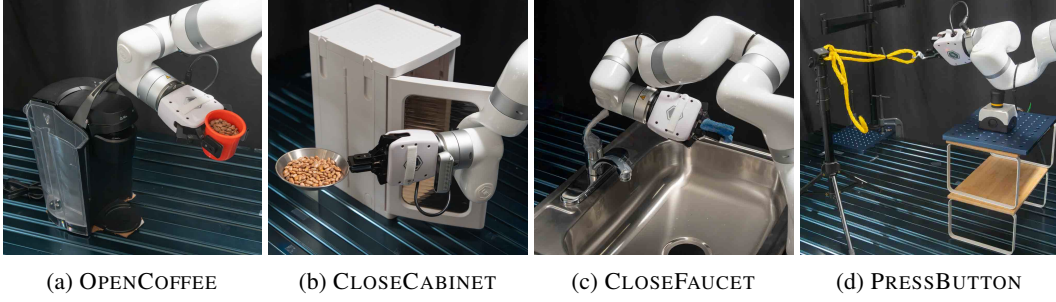


Figure 2: **Evaluation tasks.** Each task pairs a secondary whole-arm contact action with a primary end-effector task that must hold throughout: cup upright, plate level, gripper above the sink, gripper tethered by a rope.

Surface-contact inverse kinematics. For each (waypoint t , candidate anchor $\mathbf{u}$) pair, we solve a surface-contact IK problem that drives the chosen link to track $(\mathbf{p}_t^*, \mathbf{n}_t^*)$ under the cost

$$J_{\text{contact}}(q, \mathbf{u}; \mathbf{p}^*, \mathbf{n}^*) = w_{\text{pos}} \|\mathbf{p}(q, \mathbf{u}) - \mathbf{p}^*\|^2 + w_{\text{align}} [\cos \theta_{\text{dead}} - \langle \mathbf{n}(q, \mathbf{u}), \mathbf{n}^* \rangle]_+, \quad (4)$$

combining a quadratic position residual with a hinge alignment cost on the surface normal. The hinge zeros the alignment penalty whenever the angular error is within the deadzone θ_{dead} , so IK can trade a small tilt of the normal for position accuracy. Standard joint-limit, self-collision, scene-collision, and $\{c_k\}$ soft costs are added to the IK objective [54]; solutions that exceed a tolerance τ_k or collide with the scene are discarded, leaving admissible $(t, \mathbf{u}, q)$ triples for beam expansion.

Beam search and trajectory assembly. Per-waypoint IK yields admissible $(\mathbf{u}, q)$ candidates independently at each t , but an executable trajectory needs adjacent waypoints to agree on joint configuration and to stay close on the contact surface. Exhaustive enumeration over H -step sequences is intractable, so we assemble the trajectory with beam search of width B , configured as in Appendix A.5. Each partial sequence $(\mathbf{u}_{1:t}, q_{1:t})$ is extended by each admissible anchor at $t+1$, scored by the step cost

$$J_{\text{step}}(q_t, \mathbf{u}_t \rightarrow q_{t+1}, \mathbf{u}_{t+1}) = w_{\Delta q} \|q_{t+1} - q_t\| + w_{\text{slide}} d_{\text{slide}} + w_{\text{manip}}/\mu(q_{t+1}), \quad (5)$$

where $d_{\text{slide}} = \|\mathbf{u}_{t+1} - \mathbf{u}_t\|$ is the link-local displacement of the surface anchor between t and $t+1$ and μ is the Yoshikawa manipulability [55]; we keep the top- B extensions until $t = H$. The search runs in parallel for each candidate link $\ell \in \mathcal{L}$ and the results are pooled, so the optimizer selects the link whose trajectory scores best for the given $\mathcal{T}^*$ and $\{C_k\}$ rather than committing to one in advance. For each pooled candidate we sample a pre-contact pose by backing off along the contact normal at $t = 1$ within a small offset cone, then plan a free-space approach to it from the home pose under $\{C_k\}$. We rank the resulting (link, contact, approach) triples by feasibility margin, breaking ties by accumulated slide $\sum_t d_{\text{slide}}$ and then by joint path length, and execute the top one.

3.5 Inference and execution

At inference, the stages of Figure 1 run in sequence on a single observation $O = (I, d_{\text{task}})$; Algorithm 1 gives the pseudocode. The keypoint matcher takes K_{obs} from the highest-scoring camera view, the policy samples $\mathcal{T}^* \sim \pi_{\theta}(\cdot \mid K_{\text{obs}})$, and the constraint pipeline produces $\{C_k\}$ from the same observation. The surface-contact optimizer (Section 3.4) then returns the top-ranked joint trajectory with its free-space approach, which are concatenated and streamed to the robot.

4 Experiments

We design our experiments to answer three questions. First, does WACL outperform a strong link-pose baseline on real-world whole-arm tasks, and does the gap widen under end-effector constraints (Section 4.2)? Second, does it generalize to additional constraints without retraining, and to a different robot (Section 4.3)? Third, which design choices are responsible for the gain (Section 4.4)?

4.1 Experimental setup

Hardware and workspace. All experiments run on a 7-DoF UFactory xArm 7 observed by two calibrated Intel RealSense D435 RGB-D cameras, with planning on one NVIDIA RTX PRO 6000

Task	Unconstrained			Constrained			End-effector constraint
	KALM	WACL (Ours)	WACL (FR3)	KALM	WACL (Ours)	WACL (FR3)	
OPENCOFFEE	4/10	6/10	5/10	2/10	7/10	4/10	Hold cup upright
CLOSECABINET	7/10	7/10	8/10	5/10	8/10	7/10	Hold plate level
CLOSEFAUCET	6/10	8/10	8/10	3/10	7/10	8/10	Keep gripper above sink
PRESSBUTTON	6/10	9/10	9/10	3/10	9/10	8/10	Stay tethered by rope
Aggregate	58%	75%	75%	33%	78%	68%	

Table 1: **Real-world results on four whole-arm tasks.** We evaluate the KALM baseline and our method, WACL, on four whole-arm tasks under both unconstrained and constrained settings, each over 10 rollouts with 10 demonstrations per task (number of successes shown). KALM and WACL (Ours) run on the xArm 7; WACL (FR3) runs zero-shot on a Franka FR3 with only the optimizer’s robot model changed. Under constraints, WACL more than doubles KALM’s aggregate success (78% vs. 33%).

GPU, described in Appendix B.1. The optimizer searches in parallel over the three distal contact-capable links of the xArm, links 4–6, whose collision meshes form the contact surface $\mathcal{M}$ (Figure 3).

Tasks and end-effector constraints. We evaluate four whole-arm tasks, each constructed so that a primary end-effector task must hold throughout a secondary contact action. We instantiate two primitives from Section 3.3: an AXISCONE keeps a link axis within a tolerance angle of a target direction, and a POSITIONREGION keeps a link position inside a 3D region, here a sphere or a box; the four tasks are (Figure 2):

- OPENCOFFEE: lift the handle of a coffee machine while the gripper holds a cup of water (AXISCONE: cup axis aligned with world up).
- CLOSECABINET: push a cabinet door closed while the gripper holds a plate (AXISCONE: plate normal aligned with world up).
- CLOSEFAUCET: turn a faucet handle off while the gripper holds a wet scouring pad (POSITIONREGION: gripper inside a box above the sink).
- PRESSBUTTON: press a button while the gripper is tethered to a fixed point by a short rope (POSITIONREGION: gripper inside a sphere around the rope anchor).

The VLM generates these constraints at inference, and Appendix B.2 audits their accuracy.

Baseline. KALM [16] is a keypoint-conditioned diffusion-model method that regresses an $SE(3)$ end-effector trajectory from semantic scene keypoints, originally designed for prehensile manipulation. To evaluate KALM on whole-arm contact, we train it to regress the $SE(3)$ pose of one contact link, chosen per task by the teleoperator, instead of the gripper pose; KALM’s distillation, architecture, and training objective are unchanged. We realize the predicted link-pose trajectory on the robot through the same downstream trajectory optimizer used by WACL, with one modification: because the contact target is a 6-DoF link pose rather than a position-and-normal pair, the surface-anchor search is disabled, while constraint handling and beam search are kept identical. This isolates the comparison to the action representation, link-pose trajectory versus surface-contact trajectory, together with the demonstrations each representation requires, rather than to downstream planning. The whole-arm systems of Section 2 are not suitable baselines here: they need an object model [28, 29, 30] or a simulator and a tactile skin [27], or learn joint-space policies that take no end-effector constraint [10]. We compare against KALM because WACL builds on its keypoint pipeline, so the two differ only in what they regress from the same keypoints; controls that replace the learned trajectory are in Section 4.4.

Demonstrations. For WACL, we collect 10 bare-hand human demonstrations per task using the multi-camera capture pipeline of Section 3.2.1. For the baseline, we instead collect 10 teleoperated demonstrations per task using a GELLO [9] joint-space leader-follower controller, which enables the teleoperator to collect whole-arm demonstrations. They span varied contact poses and surface regions on the chosen link, so the baseline sees comparable contact diversity. The two methods thus learn from different demonstration modalities, and we do not separate the two effects: a bare-hand demonstration cannot define the 6-DoF link pose that KALM regresses.



Figure 3: **Contact links.** Collision meshes of the xArm 7 **forearm** (link 4), **wrist yaw** (link 5), and **wrist pitch** (link 6) form the contact surface $\mathcal{M}$.

Variant	What changes	Task	Success	
			Variant	Full WACL
+ second constraint	cup-upright AXISCONE added	CLOSEFAUCET	7/10	7/10
w/o sliding anchor	anchor fixed at first waypoint	OPENCOFFEE	3/10	7/10
w/o link search	link fixed to link 5	PRESSBUTTON	6/10	9/10
	link fixed to link 6	PRESSBUTTON	7/10	9/10
geometric primitive	hand-specified arc as trajectory	OPENCOFFEE	4/10	7/10
	hand-specified line as trajectory	PRESSBUTTON	7/10	9/10
registered replay	ICP-registered demonstration	OPENCOFFEE	6/10	7/10

Table 2: **Generalization, ablations, and controls.** Each variant runs 10 rollouts on one constrained task, on the same placements as the full model. A second end-effector constraint leaves success unchanged, while removing an optimizer component or replacing the learned trajectory lowers it.

Protocol and success criterion. Each (task, method, setting) cell is evaluated over 10 rollouts on the physical robot. Between rollouts we randomize the task-object placement and the robot’s home configuration; placements are physically marked and reset so that both methods see the same set, and every placement is kinematically reachable rather than filtered by either planner. The planner gates every candidate trajectory against the active tolerances $\{\tau_k\}$ (Section 3.3), so a rollout either executes a constraint-satisfying trajectory or returns no plan. We count a rollout as a failure if it returns no plan or does not reach the task subgoal. The *unconstrained* setting drops the end-effector constraint from the planner, leaving only the contact action; the *constrained* setting enforces it.

4.2 Constrained whole-arm manipulation on a 7-DoF xArm

Table 1 reports the main results: under the unconstrained setting, WACL succeeds on most rollouts of every task and is on par with or above KALM throughout, while KALM is less reliable on the harder tasks (e.g., 4/10 on OPENCOFFEE), consistent with whole-arm contact tolerating small targeting errors once a large contact patch is established. Under constraints, KALM drops on every task, to half its unconstrained rate on three of four, while WACL is essentially unaffected. The asymmetry is structural: KALM commits to a link-pose trajectory fixed in the demonstrator’s embodiment and leaves IK no freedom once a constraint makes it infeasible, whereas WACL searches jointly over link, anchor, and joints and reaches the same target through whichever link and anchor satisfy the constraint. Aggregated over the four tasks, the constrained gap is significant ($p = 4.0 \times 10^{-5}$) and the unconstrained gap is not ($p = 0.12$); Appendix B.3 reports the tests. The optimizer returned a complete plan on 69/74 logged planning attempts in the constrained sessions; Appendix B.4 tallies the failure modes. Table 5 shows the same pattern when both planners replay logged scenes offline.

4.3 Generalization to new end-effector constraints and embodiments

Multiple end-effector constraints. The constraint pipeline of Section 3.3 is regenerated from the task description at every inference call, so adding a second end-effector constraint is a change of input rather than a change of model. We stress-test this property on CLOSEFAUCET by swapping the pad for a cup that can spill, which layers an AXISCONE cup-upright requirement on the POSITIONREGION above-sink constraint the task already carries. WACL preserves its single-constraint success rate under the tightened two-constraint set, 7/10 with one constraint and with two (Table 2). Additional constraints enter the optimizer as soft costs paired with hard tolerances, tightening the feasible set without disturbing the contact objective or the policy that produced it.

Zero-shot cross-embodiment transfer. Because the robot’s kinematics enter only in the optimizer, the same keypoint policy, reference contact trajectories, and task descriptions run on a 7-DoF Franka FR3 with a Robotiq gripper after changing only the optimizer’s robot model (Table 1, WACL (FR3) columns). The FR3 reaches 75% unconstrained and 68% constrained; the constrained gap to the xArm is concentrated on OPENCOFFEE (4/10), where the FR3’s thicker links struggle to thread the slim handle, while the other tasks transfer almost unchanged, as analyzed in Appendix B.6.

4.4 Ablation study

Table 2 removes one component at a time to show which ones drive WACL’s constrained performance.

Sliding anchor. To test whether the contact must slide along the link, we pin the surface anchor $\mathbf{u}_t \equiv \mathbf{u}_1$ to its first-waypoint value, which reduces surface-contact IK to fixed-anchor IK. Success on OPENCOFFEE drops from 7/10 to 3/10: the fixed anchor becomes IK-infeasible mid-trajectory under the cup-upright constraint, which a sliding anchor avoids; Appendix B.7 traces the failures.

Link search. To test whether the optimizer must choose the contact link, we pin it to link 5 or to link 6 on PRESSBUTTON. Link 5 alone gives 6/10 and link 6 alone 7/10, against 9/10 with the search; the two links succeed on largely different placements (two only with link 5, three only with link 6), so each covers placements the other misses, and the search recovers both.

Geometric primitive. To test whether a learned trajectory is needed, we replace the policy with a hand-specified geometric primitive registered to the scene, a line for PRESSBUTTON and an arc for OPENCOFFEE. The geometric primitive stays close to the policy on the translational press (7/10 vs. 9/10) but drops from 7/10 to 4/10 on OPENCOFFEE, where the capsule carriage translates while the lid rotates less than the handle, so no circular arc fits. The policy instead learns this coupled motion from the demonstrations; Appendix B.7 details the primitive.

Registered replay. To test what the policy adds over the demonstrations themselves, we replay one demonstration registered to the scene by iterative closest point (ICP). Replay reaches 6/10 on OPENCOFFEE, so the demonstrated motion does carry the task; the policy reaches 7/10 without registration or per-task parameters and proposes several candidates per scene instead of one.

5 Conclusion

We presented *Whole-Arm Contact Learning* (WACL), a framework that performs a whole-arm contact action without disturbing a primary end-effector task. Across four real-world tasks on a 7-DoF xArm, WACL more than doubles the constrained success rate of a strong link-pose baseline (78% vs. 33%), absorbs a second simultaneous constraint without retraining, and transfers zero-shot to a second 7-DoF arm by changing only the optimizer’s robot model. Decoupling what the task requires from how an arm achieves it lets these skills carry over to new constraints and a second arm.

6 Limitations

WACL plans contact geometrically: it tracks a contact position and normal and models neither force nor object dynamics, so a feasible plan can still slip or under- or over-press, as the failure cases in Appendix B.4 show. That contact is also a single point on a single link; a second simultaneous contact would have its spacing fixed by the arm’s link lengths, so the reference trajectory would no longer be robot-agnostic. The constraints on the other links come from a vocabulary of two primitive types, AXISCONE and POSITIONREGION; a requirement neither expresses cannot be enforced, although each primitive is a short violation function that a VLM could write as code. Keypoints, trajectory, and constraints are all computed from one RGB-D frame and run open-loop over a short horizon of at most 8 waypoints, and one inference with a single candidate plan takes 34.6 s, most of it in two hosted VLM calls, as the per-stage breakdown in Appendix A.7 shows. The optimizer’s own share is small, but its anchor search grows with the number of contact-capable links; Appendix C lists further limitations of the evaluation. Coupling the planner with force or impedance control, enriching the contact representation with multi-link and line- or patch-contact primitives, and closing the loop with tactile or visual replanning remain to be explored.

7 Acknowledgments

The authors thank the USC Center for Advanced Research Computing (CARC) for providing compute resources. This work was supported by the Office of Naval Research (ONR) Young Investigator Program (YIP) under grant number N00014-26-1-2473. This research was also supported by the NVIDIA Academic Grant Program using an RTX PRO 6000 Blackwell Max-Q Workstation Edition.

References

- [1] B. S. Eberman. Whole-arm manipulation: Kinematics and control. Master’s thesis, Massachusetts Institute of Technology, 1989.
- [2] B. Zitkovich, T. Yu, S. Xu, P. Xu, T. Xiao, F. Xia, J. Wu, P. Wohlhart, S. Welker, A. Wahid, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, 2023.
- [3] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi, et al. OpenVLA: An open-source vision-language-action model. In *Conference on Robot Learning*, 2024.
- [4] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, et al. π_0 : A vision-language-action flow model for general robot control. In *Robotics: Science and Systems*, 2025.
- [5] K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, M. Y. Galliker, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. In *Conference on Robot Learning*, 2025.
- [6] J. Bjorck, F. Castañeda, N. Cherniadev, X. Da, R. Ding, L. Fan, Y. Fang, D. Fox, F. Hu, S. Huang, et al. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv:2503.14734*, 2025.
- [7] Gemini Robotics Team, S. Abeyruwan, J. Ainslie, J.-B. Alayrac, M. G. Arenas, T. Armstrong, A. Balakrishna, R. Baruch, M. Bauza, M. Blokzijl, et al. Gemini Robotics: Bringing AI into the physical world. *arXiv:2503.20020*, 2025.
- [8] J. Lee, J. Duan, H. Fang, Y. Deng, S. Liu, B. Li, B. Fang, J. Zhang, Y. R. Wang, S. Lee, et al. MolmoAct: Action reasoning models that can reason in space. *arXiv:2508.07917*, 2025.
- [9] P. Wu, Y. Shentu, Z. Yi, X. Lin, and P. Abbeel. GELLO: A general, low-cost, and intuitive teleoperation framework for robot manipulators. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2024.
- [10] H. Fang, H.-S. Fang, Y. Wang, J. Ren, J. Chen, R. Zhang, W. Wang, and C. Lu. AirExo: Low-cost exoskeletons for learning whole-arm manipulation in the wild. In *IEEE International Conference on Robotics and Automation*, 2024.
- [11] M. Lepert, J. Fang, and J. Bohg. Phantom: Training robots without robots using only human videos. In *Conference on Robot Learning*, 2025.
- [12] S. Kareer, D. Patel, R. Punamiya, P. Mathur, S. Cheng, C. Wang, J. Hoffman, and D. Xu. EgoMimic: Scaling imitation learning via egocentric video. In *IEEE International Conference on Robotics and Automation*, 2025.
- [13] B.-R. Huang, C.-K. Yang, C.-M. Lai, D.-J. Wu, and S.-H. Sun. Diffusion imitation from observation. In *Advances in Neural Information Processing Systems*, 2024.
- [14] M. Hong, A. Liang, K. Kim, H. Rajaprakash, J. Thomason, E. Bıyık, and J. Zhang. HAND me the data: Fast robot adaptation via hand path retrieval. In *IEEE International Conference on Robotics and Automation*, 2026.
- [15] S. Sontakke, J. Zhang, S. Arnold, K. Pertsch, E. Bıyık, D. Sadigh, C. Finn, and L. Itti. RoboCLIP: One demonstration is enough to learn robot policies. In *Advances in Neural Information Processing Systems*, 2023.
- [16] X. Fang, B.-R. Huang, J. Mao, J. Shone, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling. KALM: Keypoint abstraction using large models for object-relative imitation learning. In *IEEE International Conference on Robotics and Automation*, 2025.
- [17] H. Ha, Y. Gao, Z. Fu, J. Tan, and S. Song. UMI-on-Legs: Making manipulation policies mobile with manipulation-centric whole-body controllers. In *Conference on Robot Learning*, 2024.

- [18] Z. Fu, X. Cheng, and D. Pathak. Deep whole-body control: Learning a unified policy for manipulation and locomotion. In *Conference on Robot Learning*, 2022.
- [19] S. Jeon, M. Jung, S. Choi, B. Kim, and J. Hwangbo. Learning whole-body manipulation for quadrupedal robot. *IEEE Robotics and Automation Letters*, 9(1):699–706, 2024.
- [20] X. Cheng, Y. Ji, J. Chen, R. Yang, G. Yang, and X. Wang. Expressive whole-body control for humanoid robots. In *Robotics: Science and Systems*, 2024.
- [21] S. Wistreich, B. Shi, S. Tian, S. Clarke, M. Nath, C. Xu, Z. Bao, and J. Wu. DexSkin: High-coverage conformable robotic skin for learning contact-rich manipulation. In *Conference on Robot Learning*, 2025.
- [22] Y. Luo, Y. Li, P. Sharma, W. Shou, K. Wu, M. Foshey, B. Li, T. Palacios, A. Torralba, and W. Matusik. Learning human–environment interactions using conformal tactile textiles. *Nature Electronics*, 4(3):193–201, 2021.
- [23] B. Xu, L. Zhong, G. Zhang, X. Liang, D. Virtue, R. Madan, and T. Bhattacharjee. CushSense: Soft, stretchable, and comfortable tactile-sensing skin for physical human-robot interaction. In *IEEE International Conference on Robotics and Automation*, 2024.
- [24] Z. Si, T. C. Yu, K. Morozov, J. McCann, and W. Yuan. RobotSweater: Scalable, generalizable, and customizable machine-knitted tactile skins for robots. In *IEEE International Conference on Robotics and Automation*, 2023.
- [25] H. Fang, C. Wang, Y. Wang, J. Chen, S. Xia, J. Lv, Z. He, X. Yi, Y. Guo, X. Zhan, et al. AirExo-2: Scaling up generalizable robotic imitation learning with low-cost exoskeletons. In *Conference on Robot Learning*, 2025.
- [26] H.-S. Fang, B. Romero, Y. Xie, A. Hu, B.-R. Huang, J. Alvarez, M. Kim, G. Margolis, K. Anbarasu, M. Tomizuka, et al. DEXOP: A device for robotic transfer of dexterous human manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2026.
- [27] J. A. Barreiros, A. Ö. Önl, M. Zhang, S. Creasey, A. Goncalves, A. Beaulieu, A. Bhat, K. M. Tsui, and A. Alspach. Learning contact-rich whole-body manipulation with example-guided reinforcement learning. *Science Robotics*, 10(105):eads6790, 2025.
- [28] T. Pang, H. J. T. Suh, L. Yang, and R. Tedrake. Global planning for contact-rich manipulation via local smoothing of quasi-dynamic contact models. *IEEE Transactions on Robotics*, 39(6):4691–4711, 2023.
- [29] T. Xue, Y. Zhang, A. Razmjoo, and S. Calinon. Monte Carlo tree search with tensor factorization for optimization problems in robotics. *arXiv:2507.04949*, 2025.
- [30] V. Levé, J. Moura, N. Saito, S. Tonneau, and S. Vijayakumar. Explicit contact optimization in whole-body contact-rich manipulation. In *IEEE-RAS International Conference on Humanoid Robots*, 2024.
- [31] X. Li, T. Zhao, X. Zhu, J. Wang, T. Pang, and K. Fang. Planning-guided diffusion policy learning for generalizable contact-rich bimanual manipulation. *arXiv:2412.02676*, 2024.
- [32] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei. VoxPoser: Composable 3D value maps for robotic manipulation with language models. In *Conference on Robot Learning*, 2023.
- [33] K. Fang, F. Liu, P. Abbeel, and S. Levine. MOKA: Open-world robotic manipulation through mark-based visual prompting. In *Robotics: Science and Systems*, 2024.
- [34] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as Policies: Language model programs for embodied control. In *IEEE International Conference on Robotics and Automation*, 2023.
- [35] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar. Eureka: Human-level reward design via coding large language models. In *International Conference on Learning Representations*, 2024.

- [36] Y. Wang, Z. Sun, J. Zhang, Z. Xian, E. Bıyık, D. Held, and Z. Erickson. RL-VLM-F: Reinforcement learning from vision language foundation model feedback. In *International Conference on Machine Learning*, 2024.
- [37] M. Atasever, K. Azbijari, C. Bakirci, B.-R. Huang, T. Izdas, Z. Shahrooei, R. Yang, E. Bıyık, and J. V. Deshmukh. Video2STL: Grounding VLM-generated temporal specifications for robot learning. *arXiv:2609.37519*, 2026.
- [38] W. Huang, C. Wang, Y. Li, R. Zhang, and L. Fei-Fei. ReKep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation. In *Conference on Robot Learning*, 2024.
- [39] H. Huang, F. Lin, Y. Hu, S. Wang, and Y. Gao. CoPa: General robotic manipulation through spatial constraints of parts with foundation models. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2024.
- [40] Y. Chen, J. Arkin, C. Dawson, Y. Zhang, N. Roy, and C. Fan. AutoTAMP: Autoregressive task and motion planning with LLMs as translators and checkers. In *IEEE International Conference on Robotics and Automation*, 2024.
- [41] Y. Ling, K. Owalekar, O. Adesanya, E. Bıyık, and D. Seita. IMPACT: Intelligent motion planning with acceptable contact trajectories via vision-language models. In *IEEE International Conference on Robotics and Automation*, 2026.
- [42] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, et al. Do as I can, not as I say: Grounding language in robotic affordances. In *Conference on Robot Learning*, 2022.
- [43] D. Driess, F. Xia, M. S. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, et al. PaLM-E: An embodied multimodal language model. In *International Conference on Machine Learning*, 2023.
- [44] J. Zhang, M. Memmel, K. Kim, D. Fox, J. Thomason, F. Ramos, E. Bıyık, A. Gupta, and A. Li. PEEK: Guiding and minimal image representations for zero-shot generalization of robot manipulation policies. In *IEEE International Conference on Robotics and Automation*, 2026.
- [45] N. Carion, L. Gustafson, Y.-T. Hu, S. Debnath, R. Hu, D. Suris, C. Ryali, K. V. Alwala, H. Khedr, A. Huang, et al. SAM 3: Segment anything with concepts. In *International Conference on Learning Representations*, 2026.
- [46] J. Romero, D. Tzionas, and M. J. Black. Embodied hands: Modeling and capturing hands and bodies together. *ACM Transactions on Graphics*, 36(6):245:1–245:17, 2017.
- [47] Y. Ye, Y. Feng, O. Taheri, H. Feng, S. Tulsiani, and M. J. Black. Predicting 4D hand trajectory from monocular videos. In *International Conference on 3D Vision*, 2026.
- [48] R. I. Hartley and P. Sturm. Triangulation. *Computer Vision and Image Understanding*, 68(2): 146–157, 1997.
- [49] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, et al. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*, 2024.
- [50] R. B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (FPFH) for 3D registration. In *IEEE International Conference on Robotics and Automation*, 2009.
- [51] M. Janner, Y. Du, J. B. Tenenbaum, and S. Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning*, 2022.
- [52] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. V. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, 2022.

- [53] Y. Eldar, M. Lindenbaum, M. Porat, and Y. Y. Zeevi. The farthest point strategy for progressive image sampling. *IEEE Transactions on Image Processing*, 6(9):1305–1315, 1997.
- [54] B. Sundaralingam, S. K. S. Hari, A. Fishman, C. Garrett, K. Van Wyk, V. Blukis, A. Millane, H. Oleynikova, A. Handa, F. Ramos, et al. cuRobo: Parallelized collision-free robot motion generation. In *IEEE International Conference on Robotics and Automation*, 2023.
- [55] T. Yoshikawa. Manipulability of robotic mechanisms. *International Journal of Robotics Research*, 4(2):3–9, 1985.
- [56] D. Coleman, I. Sucan, S. Chitta, and N. Correll. Reducing the barrier to entry of complex robotic software: A MoveIt! case study. *Journal of Software Engineering for Robotics*, 5(1): 3–16, 2014.
- [57] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.

Appendix

This supplement collects the details needed to reproduce WACL and to verify the claims in the main paper, in the order the main text uses them. Appendix A details the components of Section 3: the demonstration data pipeline, the keypoint-conditioned policy, the constraint primitives and their generation, the surface-contact optimizer, and the inference pipeline with its runtime. Appendix B describes the real-robot setup and reports the constraint-generation audit, confidence intervals, failure cases, the paired offline replay, cross-embodiment transfer, and the ablation and control details. Appendix C lists further limitations.

A Method details

A.1 Demonstration data pipeline

Multi-view capture. A human performs each task bare-handed in front of $C = 3$ calibrated RGB-D cameras, the two scene cameras of Appendix B.1 and the arm’s wrist camera, held static during capture, extrinsically calibrated to a common world frame. We record synchronized RGB-D streams from every camera at 10 Hz. We collect 10 demonstrations per task, deliberately varying the contact location so the learned policy sees a diversity of contact poses.

Multi-view fusion. Before fusing the per-view MANO tracks of Section 3.2.1, we shift each view’s mesh along its optical axis so that its visible vertices match the sensor depth. We then fuse the contact fingertip vertex across the C views by per-point weighted-midpoint ray triangulation [48]: each view contributes a back-projected ray through its fingertip estimate, and the fused point minimizes the weighted sum of squared distances to these rays. Each view’s weight is the silhouette IoU of its MANO mesh reprojected into every view against that view’s SAM3 hand mask, summed over views, so views whose hand estimate is geometrically consistent dominate the fusion.

A.2 Keypoint-conditioned policy

The KALM [16] policy of Section 3.2.2 is trained unchanged in architecture on our (K_{obs}, T^*) pairs with $n_{\text{kp}} = 8$ keypoints, embedding dimension 256, trajectory length 16, 100 diffusion steps, 100k training iterations, batch size 512, and AdamW with learning rate 10^{-4} and weight decay 5×10^{-4} ; the KALM baseline uses the same settings. When several camera views are available, we keep the trajectory from the view whose keypoints have the highest mean feature similarity to the reference.

A.3 Constraint primitives

Section 3.3 defines the violation c_k and the tolerance τ_k that every primitive exposes. Below we specify the two primitives we use, whose per-task parameters Table 3 lists, and the two implementation choices that definition leaves open: the penalty each violation computes and how its tolerance is enforced. We add each violation as a soft cost to the trajectory optimizer as a native cuRobo constraint, scaled by the quantity it penalizes, radians for AXISCONE and meters for POSITIONREGION, so the two weights are tuned independently. We enforce the tolerance with a *headroom* gate that maps

Task	Primary end-effector requirement	Primitive	Deployed parameters
OPENCOFFEE	hold cup upright	AXISCONE	TCP x -axis within 25° of world $+z$
CLOSECABINET	hold plate level	AXISCONE	TCP y -axis within 25° of world $+z$
CLOSEFAUCET	keep gripper above sink	POSITIONREGION	TCP inside a $0.50 \times 0.40 \times 1.40$ m box over the sink
PRESSBUTTON	stay tethered by rope	POSITIONREGION	TCP inside a sphere of radius 0.20 m at the anchor
CLOSEFAUCET, two constraints	above the sink, cup upright	both	$0.50 \times 0.40 \times 1.00$ m box, plus TCP x -axis within 25° of $+z$

Table 3: **Deployed end-effector constraints.** The primitive and the parameters that ran on the robot for each task; the stated angle or region boundary is the primitive’s parameter, enforced as a hard limit ($\tau_k = 0$); TCP is the tool center point.

every constraint onto a common normalized scale. At each waypoint it measures the fraction of the primitive’s budget consumed: $\gamma(q)/\alpha$ for AXISCONE, or the fraction of the region extent used for POSITIONREGION. It discards the trajectory as soon as any waypoint exceeds a cutoff; all reported experiments use a cutoff of 1.0, i.e., $\tau_k = 0$, so the stated bound is the safety specification itself. It also defines the ranking’s feasibility margin, the smallest budget left over constraints and waypoints.

AXISCONE. With parameters $\phi_k = (\ell_k, \mathbf{a}, \mathbf{a}_w, \alpha)$, a constrained link ℓ_k , a link-local axis $\mathbf{a}$, a target world axis $\mathbf{a}_w$, and a half-angle α , the primitive penalizes the angle by which the world-frame axis $R_{\ell_k}(q) \mathbf{a}$ of Equation (2) leaves the cone around $\mathbf{a}_w$,

$$c_k(q) = [\gamma(q) - \alpha]_+, \quad \gamma(q) = \arccos \langle R_{\ell_k}(q) \mathbf{a}, \mathbf{a}_w \rangle. \quad (6)$$

The penalty is exactly zero inside the cone and grows linearly in the violation angle outside it. We penalize the angle γ directly rather than its cosine. A cosine-domain hinge has a restoring force proportional to $\sin \gamma$, which is small at the boundary of a tight cone, so small violations are corrected only weakly. The angle-space L1 penalty instead keeps a constant restoring force per radian, matching the angular quantity the headroom gate checks.

POSITIONREGION. With parameters $\phi_k = (\ell_k, \mathcal{R})$, a constrained link ℓ_k and a convex region $\mathcal{R}$, a sphere or an axis-aligned box in our tasks, the primitive penalizes the exterior distance of the link position $t_{\ell_k}(q)$ of Equation (2) from $\mathcal{R}$,

$$c_k(q) = \text{dist}(t_{\ell_k}(q), \mathcal{R}), \quad (7)$$

which is zero inside $\mathcal{R}$ and grows linearly with the distance outside. We evaluate the exterior distance with a pseudo-Huber softened norm, $\sqrt{\|\cdot\|^2 + \epsilon} - \sqrt{\epsilon}$ with $\epsilon = 10^{-8}$, so the gradient stays finite at the region boundary.

A.4 Safety-aware constraint generation

Two-call pipeline. The first of the two GPT-5.5 calls of Section 3.3, shown in Figure 4, emits for each physical entity it names an ensemble of short SAM3 text prompts ordered from a bare noun to a context-rich phrase. SAM3 segments each entity in every calibrated view, and the back-projected point clouds are associated across views before the oriented bounding box is fit. The second call, shown in Figure 5, follows generic rules that derive the constrained axis from the grasp geometry, keep the constraint set minimal, and add a margin to every region, and it covers latent risks such as spilling, dripping, shedding, contamination, fragility, or loss of task utility.

Vocabulary. We fix the primitive vocabulary to AXISCONE and POSITIONREGION for all reported experiments, but the constraint-instantiation prompt accepts new primitive types without changing any learned component, so richer VLM-driven constraint generation, broader vocabularies, and tighter scene grounding are a natural direction for further exploration.

A.5 Surface-contact trajectory optimizer

We build WACL on top of cuRobo [54], whose GPU-batched inverse-kinematics and trajectory-optimization kernels we use to evaluate the full per-link, per-anchor candidate fan-out in parallel. The surface-contact cost of Equation (4), the constraint primitives of Appendix A.3, and the beam-search assembly are our additions on top of this solver. Below we describe the components whose behavior is not obvious from the main text; all tasks share the cost weights $w_{\text{pos}} = 10^4$, $w_{\text{align}} = 2 \times 10^3$, $w_{\Delta q} = 1$, $w_{\text{slide}} = 10 \text{ m}^{-1}$, and $w_{\text{manip}} = 2 \times 10^{-3}$.

Reachable anchor set. The anchor pruning of Section 3.4 runs before any IK; an anchor counts as occluded when an adjacent link blocks its outward normal within 0.05 m, and as buried when it lies inside a neighboring link’s signed-distance volume with margin -0.002 m . At each sliding step we re-sample anchors only within a small Euclidean radius (0.03 m) of the parent anchor, so the search refines the contact locally rather than re-scanning the entire link surface every step.

Collision buffers. Whole-arm contact requires the active link to make contact with the environment, which a standard collision check forbids. We resolve this by shrinking the contact link’s collision spheres with a negative buffer during the sliding stage, admitting controlled surface penetration, while every other link keeps a small positive clearance. During the free-space approach and the pre-contact pose we use normal positive buffers, keeping the standard collision-avoidance behavior. The optimizer plans the approach with cuRobo’s gradient-based trajectory optimization [54], which can stall in a local minimum near obstacles; when it fails, MoveIt [56] plans the approach under the same path constraints with the sampling-based RRTConnect planner, which searches a graph of collision-free configurations instead of descending a cost.

Prompt 1: Scene decomposition

VLM call 1, input: RGB frame and text

You are a visual grounding assistant. Your task is to decompose a task description into discrete physical entities and generate a diverse ensemble of short text prompts for each one. These prompts will be used to segment targets via SAM3, which performs best with simple, concrete noun phrases and degrades on overly long or abstract descriptions.

You receive a single RGB image - the CURRENT FRAME of a robot's workspace.

Strict usage rules:

- CURRENT FRAME is your ONLY source of truth for the object's true colour, texture, shape, and identity.
- The robot arm may be visible in the image - do NOT include it as a target; segment only task-relevant scene objects.

You must follow this three-step process and encode each step explicitly in your JSON output:

STEP 1 — Scene Decomposition.

Read the task description and identify every distinct physical object that must be independently located in the image. Ask yourself: "How many separate things do I need to understand the spatial layout of?" For example:

- "Wiping a countertop" → countertop, sponge
- "Pushing a box onto a shelf" → box, shelf
- "Sweeping debris into a dustpan" → dustpan, debris

Include an object or surface that merely *bounds* where the gripper must remain (a catchment, support, or safe workspace) ONLY when the task explicitly requires bounding the gripper's position. Do not add a speculative "safe region" for every state-sensitive held item; listing an extra surface invites an unnecessary constraint downstream.

Write one sentence naming every object you found. This becomes "scene_decomposition".

Finally, name which single object is the CONTACT TARGET: the scene element the robot must physically act on to accomplish the task. It must be one of the objects you listed. This becomes "contact_target".

STEP 2 — Target Count.

State the exact integer number of distinct objects from Step 1. This becomes "target_count". The "targets" array MUST contain exactly this many entries.

STEP 3 — Generate Targets.

For each object, produce 2 to 5 SAM3 prompts ordered SIMPLEST to MOST COMPLEX:

1. Bare category name (e.g. "box", "shelf").
2. Generic near-synonym or shape descriptor (e.g. "wooden crate", "wall rack").
3. One key distinguishing attribute - color, material, or spatial relation (e.g. "blue box", "metal shelf").
4. Context-rich phrase only if truly needed (e.g. "blue box on the right").

Before outputting your answer, assess the scene density. Avoid generating overly broad prompts that would produce over 10 masks (e.g., "box" or "furniture" in a cluttered scene) unless absolutely necessary.

Task Description

{task_desc}

Respond ONLY with valid JSON:

```
{
  "scene_decomposition": "<one sentence listing every distinct physical object>",
  "contact_target": "<name of the object the robot must act on>",
  "target_count": <integer>,
  "targets": [
    {
      "object": "<object name>",
      "prompts": ["<prompt1 - simplest>", "<prompt2>", "..."]
    }
  ]
}
```

Figure 4: **Scene-decomposition prompt.** First VLM call of Appendix A.4. The model maps the task description to task-relevant entities, adding a bounding surface only when the task requires one, and a SAM3 prompt ensemble per entity, and names the contact target. Task-specific examples are omitted; braces mark variables.

Prompt 2: Constraint instantiation

VLM call 2, input: text only

Generate safety-aware path constraints for whole-arm manipulation.

The gripper/TCP has a primary task that must remain true at every waypoint while another arm link performs the contact action. Generate constraints only for preserving that primary TCP task, not for contact goals or collision avoidance.

Primitive choices:

- `axis_cone`: preserve orientation for upright/level/spillable/sliding/balanced held objects.
- `position_region`: preserve position when the held object/material or primary task physically implies a safe workspace, catchment, support, bounded region, or half-space. Open-vocabulary risks include dripping, leaking, shedding, contamination, heat, sharpness, fragility, tipping, dropping, tangling, or loss of task utility.
- Emit the MINIMAL set. Default to exactly ONE constraint. Add a second only if the primary task genuinely cannot be preserved by a single primitive. Do NOT add a `position_region` if an `axis_cone` already preserves the held object's state, and do not add a "safe workspace" or "catchment" region that the task does not explicitly require.
- output `[]` if no TCP path constraint is clearly needed.

Rules:

- Constrain only `link_tcp`.
- Choosing `link_axis` for an `axis_cone` (`link_axis` is a selector: `[1,0,0]`, `[0,1,0]`, or `[0,0,1]`): Reason from the GRASP GEOMETRY of the held object, not from the numbers in the TCP state. The gripper's fingers close along the local y-axis; the palm faces along the local z-axis (approach); the local x-axis is lateral across the palm. Ask: given how an object of this kind is normally held by such a gripper, which LOCAL TCP axis must stay vertical for the object to stay level / not spill / not slide off?
The TCP state below is ADVISORY ONLY: it may be a PRE-TASK / home configuration recorded before the object was picked up, so its orientation need not be representative of how the object is carried during the task. Use it as a cross-check; if the grasp-geometry reasoning disagrees with the current pose, trust the grasp-geometry reasoning.
- Use scene object OBBs to ground numeric regions, but do NOT copy an OBB extent literally: OBBs are noisy perception output and the TCP must be able to move. Expand the region by 0.10-0.15 m per side beyond the grounding object, and make sure every side of the region is at least 0.25 m so the region can contain an approach-plus-contact motion.
- Keep regions scoped to the task: the robot's task workspace is roughly 1.0 x 0.8 x 0.8 m, and a keep-in region should occupy a modest part of it (typically well under 0.3 m³). A region that spans the whole visible workspace provides no protection.
- For an "above" constraint, use a box whose x-y footprint covers the relevant support/catchment/safe workspace (plus margin) and whose z range spans the allowed height interval; the z range must start low enough to include the height at which the arm actually works, not only the space above the objects.
- A scene object listed as "NOT LOCALIZED" has no 3D estimate. Never ground a region on a NOT LOCALIZED object: ground it on the nearest localized related object, or fall back to a conservative sphere around the region the TCP must stay in.
- Round numeric values to 2 decimals.
- Output ONLY a YAML list. No prose, no markdown fence.

Current TCP State (advisory, may be a pre-task pose):

```
{tcp_state}
```

Task:

```
{task_desc}
```

Scene Objects:

```
{scene_objects}
```

YAML schema examples:

- type: `axis_cone`
link: `link_tcp`
link_axis: `<selector>`
world_axis: `<unit vector>`
angle_deg: `<degrees>`
- type: `position_region`
link: `link_tcp`
region:
 type: `box`
 center: `[<x>, <y>, <z>]`
 quaternion_wxyz: `[1.00, 0.00, 0.00, 0.00]`
 size: `[<dx>, <dy>, <dz>]`

Figure 5: **Constraint-instantiation prompt.** Second VLM call of Appendix A.4. Given the task, the tool-center-point (TCP) state from forward kinematics, and the scene oriented bounding boxes, the model derives the constrained axis from the grasp geometry, grounds regions with a margin, and emits the minimal set of typed constraint primitives as a YAML list. Task-specific examples are omitted; braces mark variables.

Surface-contact IK. Each (waypoint, anchor) pair is solved with the cost J_{contact} of Equation (4) from 16 random IK seeds. We use a hinge loss on the normal alignment rather than a full orientation cost because surface contact specifies only a position and a normal: this pins five degrees of freedom and leaves the link’s roll about the contact normal free as redundancy for the optimizer to spend on the path constraints rather than on an orientation the task never required. Its deadzone θ_{dead} is the one task-dependent cost parameter: we tighten it for tasks that need a precise contact normal (e.g., pressing a small button) and loosen it for tasks tolerant of contact-patch orientation (e.g., closing the cabinet), trading normal precision for a larger feasible set. A solution is accepted only if its position residual is within tolerance and it passes the active path-constraint gate of Appendix A.3.

Beam search. The beam search of Section 3.4 has width $B = 20$ and expands each surviving beam by 32 candidate anchors per step under the step cost of Equation (5); the first waypoint scores 640 anchors with 4 pre-contact samples each, and each later step at most 640 anchors. The step cost weighs slide far more heavily than joint motion, so the optimizer keeps the anchor stationary by default and spends slide only when the constraint set leaves no stationary-anchor extension feasible, which is the mechanism the ablation in Section 4.4 isolates. We additionally cap the per-step joint displacement $\|q_{t+1} - q_t\|$ with a maximum- Δq threshold and discard any extension that exceeds it, so beam survivors correspond to continuous, executable arm motion rather than joint-space jumps that happen to score well but would require the wrist to flip between adjacent waypoints. We downsample the reference contact trajectory $\mathcal{T}^*$ before search according to the complexity of the task: $H = 4$ for CLOSECABINET and PRESSBUTTON, and $H = 8$ for OPENCOFFEE and CLOSEFAUCET.

Why not a continuous optimization. Before the sample-and-search formulation we implemented two continuous alternatives: gradient descent over the joint trajectory with a fixed per-link contact point, and a nonlinear program over joints, per-triangle barycentric coordinates, and softmax blend weights that let the contact point move across a small patch of mesh triangles, solved with IPOPT [57]. The blended contact point and normal are not on the mesh unless the blend is peaky, so intermediate iterates are non-physical and an entropy penalty is needed to drive them back; the discreteness of the mesh and of the link choice re-enters as an outer loop over triangle patches and links, which in practice meant hundreds of multi-start solves per trajectory, most of them far from any feasible contact. The program also depended on batched per-waypoint IK for its initialization, which is the component the current optimizer keeps; replacing the continuous stage by beam search over that IK output was the change that made planning reliable. The price is greedy waypoint-by-waypoint assembly with smoothness enforced only by the step cost and per-step joint-displacement cap.

A.6 Inference pipeline

Algorithm 1 summarizes the end-to-end inference described in Section 3.5. Its first four lines are the robot-agnostic stages; the robot’s kinematics enter only in the per-link search that follows.

Algorithm 1 Inference pipeline. WACL inference on a single observation $O = (I, d_{\text{task}})$: three robot-agnostic stages feed the per-link surface-contact search.

Input: trained policy π_θ , keypoint matcher Φ_{KP} , contact links $\mathcal{L}$, per-link anchor sets $\{A_\ell\}$
Output: executable joint trajectory with a free-space approach segment

- 1: $K_{\text{obs}} \leftarrow \Phi_{\text{KP}}(I)$, KALM keypoint matching on the best-scoring view ▷ Section 3.2.2
- 2: $\mathcal{T}^* \sim \pi_\theta(\cdot \mid K_{\text{obs}})$ ▷ sample one reference contact trajectory
- 3: $\mathcal{S} \leftarrow \text{SCENE_STATE}(I, d_{\text{task}})$ ▷ VLM call 1 + SAM3 3D grounding
- 4: $\{\mathcal{C}_k\} \leftarrow \text{CONSTRAINTS}(d_{\text{task}}, \mathcal{S}, \text{FK})$ ▷ VLM call 2; Section 3.3
- 5: **for** each contact link $\ell \in \mathcal{L}$ **in parallel do**
- 6: sample pre-contact poses; IK toward $(\mathbf{p}_1^*, \mathbf{n}_1^*)$ over $\mathbf{u} \in A_\ell$; seed B beams
- 7: **for** waypoint $t = 2 \dots H$ **do**
- 8: extend each beam by IK at neighboring anchors ▷ drop collision and constraint violators
- 9: keep the top- B extensions under J_{step} ▷ Section 3.4
- 10: pool per-link trajectories; rank by feasibility margin, then slide, then path length
- 11: plan a free-space approach to the top candidate’s pre-contact pose under $\{\mathcal{C}_k\}$
- 12: **return** approach segment || contact segment

A.7 Runtime breakdown

Figure 6 reports the per-stage runtime of a representative inference with a single candidate plan. The critical path runs through constraint generation, whose three stages, scene decomposition (10.88 s), SAM3 segmentation with 3D grounding (2.76 s), and constraint instantiation (14.68 s), set the optimizer’s barrier at 28.31 s, while the keypoint policy (3.73 s) and anchor preparation (6.32 s) finish earlier and idle. The optimizer’s stages after the barrier add only 6.32 s for a total of 34.63 s, so the two hosted-VLM round-trips dominate the wall-clock while the GPU-batched planner is comparatively cheap. When a task’s end-effector constraints are known in advance, as for a task that has already been deployed, the two VLM calls can be skipped; constraints whose parameters depend on the scene, such as the region above the sink or around the rope anchor, still need the SAM3 grounding step to place them. Replanning during execution could likewise reuse the generated constraints and warm-start the optimizer from the previous solution. We did not time either mode on the robot. The executed trajectory is streamed as points 0.2 s apart and interpolated into joint commands at 250 Hz.

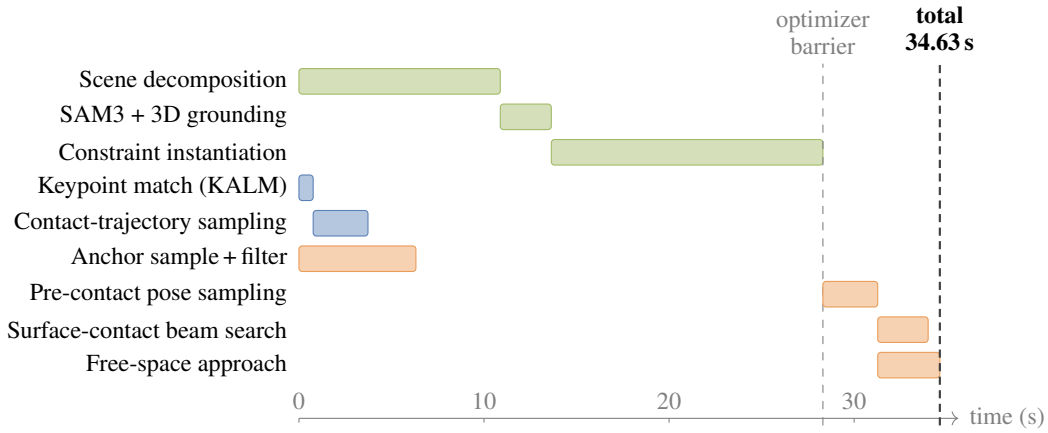


Figure 6: **Per-inference runtime breakdown.** Wall-clock per stage for a representative inference with a single candidate plan; Appendix A.7 gives the numbers. Shading groups the rows by track: constraint generation (top three rows), the keypoint policy (next two), and the optimizer (bottom four), whose anchor preparation runs concurrently with the other two tracks. The optimizer’s remaining stages start at the dashed barrier once all three tracks finish, with the beam search and the free-space approach overlapped; the right marker is the total.

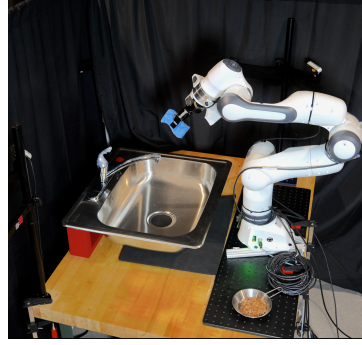
B Experiments

B.1 Real-robot setup

Our main experiments run on a 7-DoF UFactory xArm 7 mounted on a 2×1 m Vention workbench, shown in Figure 7a. Two calibrated Intel RealSense D435 RGB-D cameras are fixed to the left and right of the workspace and extrinsically calibrated to a common world frame; together with the arm’s wrist camera, they record the bare-hand human demonstrations of Appendix A.1. Planning runs on a single NVIDIA RTX PRO 6000 GPU. The cross-embodiment experiments of Appendix B.6 use a 7-DoF Franka FR3 with a Robotiq gripper on a 1.2×0.6 m workbench with a similar two-camera rig, shown in Figure 7b. The same computer is used for planning.



(a) xArm 7 workspace



(b) Franka FR3 setup

Figure 7: **Real-robot setups.** (a) the 7-DoF xArm 7 used in the main experiments; (b) the 7-DoF Franka FR3 with a Robotiq gripper used for the cross-embodiment transfer of Appendix B.6.

B.2 Constraint-generation audit

Setting	Parses	Correct type	Fully correct	Over-constrained	Looser than deployed	Re-prompts
OPENCOFFEE (AXISCONE)	20/20	20/20	20/20	0/20	0/20	0/20
CLOSECABINET (AXISCONE)	20/20	20/20	20/20	0/20	0/20	0/20
CLOSEFAUCET (POSITIONREGION)	20/20	20/20	20/20	0/20	0/20	2/20
PRESSBUTTON (POSITIONREGION)	20/20	20/20	17/20	3/20	0/20	0/20
CLOSEFAUCET, two constraints	20/20	20/20	17/20	3/20	0/20	6/20
Total	100/100	100/100	94/100	6/100	0/100	8/100

Table 4: **Constraint-generation audit.** 100 unfiltered generations (20 seeds per setting) scored against a reference constraint for each task. “Parses”: the output is a valid primitive list; “correct type”: the primitive types match the reference; “fully correct” requires the right primitive types, the right link, parameters within tolerance, acceptance of a logged successful trajectory, and rejection of adversarial probes; “over-constrained”: the output rejects that trajectory; “looser than deployed”: the output accepts it but is not fully correct; “re-prompts”: the feasibility pre-check fired. Every incorrect output is over-constrained and is converted by the planner’s hard gate into “no plan”; none is looser than the deployed constraint.

We audited the two-call generator offline on saved camera observations from real sessions: 20 seeds for each of the four tasks and for the two-constraint variant of Section 4.3, 100 generations in total, one GPT-5.5 snapshot, no cache, and no manual filtering. Each output is scored against a reference constraint for its task: primitive types, constrained link, parameters (axis within 10° , cone angle within a factor of two, region containing 95% of a logged successful trajectory), acceptance of that logged trajectory under forward kinematics, and rejection of adversarial probes that tilt or displace the end-effector. The deployed generator includes a deterministic feasibility pre-check on the raw specification (correct link, minimum region side, maximum region volume) and re-prompts once on failure; it fired in 8 of 100 generations. Table 4 reports the result: 94 fully correct, 6 over-constrained, 0 looser than deployed. The six over-constrained outputs misplace the keep-in region: three PRESSBUTTON regions are centered on the parked gripper instead of the rope anchor, and three two-constraint boxes sit above the height at which the arm works; the planner’s hard gate rejects them and returns no plan, which is counted as a task failure and is never an unsafe motion.

B.3 Confidence intervals and tests

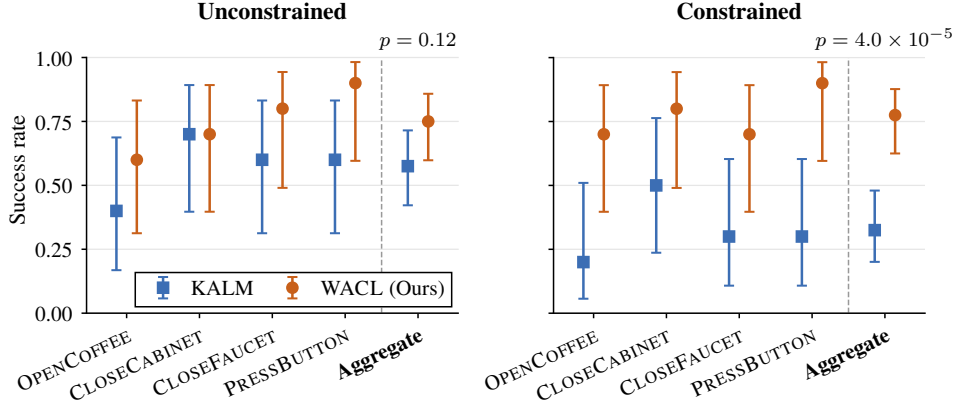


Figure 8: **Success rates with Wilson 95% intervals.** The xArm results of Table 1, 10 rollouts per task and 40 in aggregate. The aggregate constrained intervals are disjoint; the unconstrained ones overlap, consistent with the paired tests in the text; p is the exact McNemar test on the 40 placement pairs.

Figure 8 plots the xArm results of Table 1 with Wilson 95% intervals. Both methods run on the same marked placements, so the primary test is an exact McNemar test on the placement pairs, pooled over the four tasks: under constraints, WACL succeeds where KALM fails on 19 placements and the reverse holds on 1 ($p = 4.0 \times 10^{-5}$); without constraints the counts are 11 and 4 ($p = 0.12$). A paired t -test over the 40 placements gives the same conclusion ($p = 7.7 \times 10^{-6}$ and 0.07). Per task, the exact McNemar test in the constrained setting gives $p = 0.125$ (OPENCOFFEE), 0.25 (CLOSECABINET), 0.125 (CLOSEFAUCET), and 0.031 (PRESSBUTTON), without correction for the four tasks, so only the aggregate claim is made. The sliding-anchor ablation (7/10 vs. 3/10) is reported as a mechanism finding, not a significance claim.

B.4 Failure-case analysis

We report planner failures exactly, from logs, and execution failures qualitatively.

Planner failures, deployed sessions. Mining the logs of the constrained sessions behind Table 1 (four tasks, 74 planning attempts including re-plans, 148 candidate contact plans), we find that the optimizer returned a complete plan for 69/74 attempts (93%) and 124/148 candidates. All 24 failed candidates are contact-stage, and they fall into four causes: 11 sliding-path collision-gate rejections (46%, all on CLOSECABINET), 8 with no collision-free sliding transfer between waypoints (33%), 4 first-waypoint IK infeasibilities (17%), and 1 constraint-margin rejection (4%); MoveIt plans any approach the optimizer cannot, as Appendix A.5 describes.

Execution failures. In rollouts that received a feasible plan and still failed, we observed two causes. *Perception.* The keypoint matcher mislocalizes the task-relevant geometry, most often on specular or transparent surfaces (the stainless faucet), so the sampled contact trajectory targets the wrong place. It comes from the depth sensor and the keypoint matcher, either of which can be replaced by a more robust one without changing the rest of the pipeline. *Open-loop contact.* The plan is feasible but the contact slips at execution: the arm under- or over-presses because the scene is sensed once and the contact is never corrected from feedback, which is most visible on OPENCOFFEE, where the handle must be lifted through a precise arc.

B.5 Paired offline planner replay

To extend the sample without robot time, we replayed logged RGB-D observations from the deployed sessions through both planners on the same solver server, with the configuration of the Table 1 sessions restored and verified against each scene’s recorded constraint. Success here is the planner’s

own flag: every predicted contact waypoint solved by constrained IK within the deployed tolerance, with collision-checked joint paths between waypoints and the scene- and self-collision gates passed. It measures planning only. Table 5 reports the results; KALM’s rejections are explicit: constrained IK does not converge with the 6-DoF pose target inside the cone, the mechanism claimed in Section 4.2.

B.6 Cross-embodiment transfer

The Franka FR3 of Table 1 and Section 4.3, shown in Figure 7b, differs from the xArm in link geometry, contact-capable links, and end-effector frame. With only the robot model changed, the optimizer selects a different contact strategy than on the xArm on several tasks.

We attribute the constrained gap to the xArm mainly to morphology: the FR3’s links are larger and more rounded than the xArm’s, so a smaller portion of each link presents a usable contact patch and the surrounding link volume is more prone to colliding with the scene. This is most apparent on OPENCOFFEE, where the contact must thread a slim machine handle through a precise lifting arc: a thicker, rounder link both struggles to seat on the handle and sweeps a larger collision volume around it. Line- and patch-contact primitives, which Section 6 lists as future work, could narrow this gap.

B.7 Ablation and control details

Sliding anchor. In 5 of the 7 failures of the no-sliding variant on OPENCOFFEE the beam dies mid-trajectory: the fixed anchor remains feasible near the start of the contact segment but becomes IK-infeasible as the cup-upright constraint and the reference contact trajectory pull the wrist into incompatible joint configurations, at which point no admissible extension exists and beam search terminates before reaching $t = H$. Letting the anchor migrate along the link’s contact surface trades a small amount of slide for a much larger amount of joint-space slack against the constraint set.

Task	Constraint on			Constraint off		
	KALM	WACL	p	KALM	WACL	p
OPENCOFFEE	2/30	26/30	1.2×10^{-7}	7/30	30/30	2.4×10^{-7}
CLOSEFAUCET	3/30	30/30	1.5×10^{-8}	29/30	30/30	1.0

Table 5: **Paired offline planner replay.** Each row replays the same 30 logged RGB-D scenes through both planners on the same solver, with the end-effector constraint on and off, and counts complete constraint-satisfying plans; p is the exact McNemar test and nothing is executed. The OPENCOFFEE scenes are those of Table 1. Switching the constraint off helps both planners. On CLOSEFAUCET it closes the gap, while on OPENCOFFEE KALM still fails on most scenes, so its committed 6-DoF pose limits it even without the constraint.

Link search. The two pinned-link conditions on PRESSBUTTON ran on the same 10 marked placements as the paired multi-link cell, with a physical reset between conditions. Four placements were completed by both pinned links, two only by link 5, three only by link 6, and one by neither. Every pinned plan was feasible and all seven pinned failures occurred at execution, so the per-scene link choice matters for whether the executed contact succeeds, not for whether a plan exists. The union of the two pinned conditions is an oracle best-of-two; the multi-link result is 9/10.

Geometric primitive. The primitive replaces the policy by a line (PRESSBUTTON) or circular arc (OPENCOFFEE) whose anchor, axis, sweep, radius, press depth, and offsets are read off one demonstration offline and registered to each scene by ICP; no learned component contributes to the reference trajectory. This takes six parameters for PRESSBUTTON and seven for OPENCOFFEE, where the policy needs none. The demonstration also records the contact normal, so the learned trajectory presses in an effective direction, for instance upward under the coffee handle, which a primitive must be told. On OPENCOFFEE the arc failed by pushing into or slipping off the handle in 5 of 6 failures, and by an ICP misregistration in one.

Registered replay. The replay control re-poses the object-frame medoid of the 10 OPENCOFFEE demonstrations into each scene by ICP and executes it through the unchanged optimizer; registration residuals were 0.71–0.81 cm and fitness 0.65–0.94. Following a protocol fixed before the rollouts, the registered trajectory was inspected in the planner preview before each rollout with at most one

re-capture allowed on a visibly wrong fit; no re-capture was needed in any of the ten rollouts. All four failures were at execution (missing or pushing the handle); every replay produced an executable plan.

C Additional limitations

Scope of the evaluation. Robot-agnosticity is shown on the two fixed-base 7-DoF arms of Appendix B.6; arms with other kinematic structures, mobile bases, and whole-body robots are untested. Rollouts vary the object placement and the robot’s home configuration, not the object instance, so generalization of the keypoints and the contact trajectory to new object instances is not measured. No experiment pins the roll about the contact normal on its own; Table 1 and the replay of Appendix B.5 vary it only jointly with the contact link and the anchor.

Constraints and parameters. “Safety-aware” refers to the end-effector’s own task; the constraints make no guarantee about people near the robot. The alignment deadzone θ_{dead} and the trajectory horizon H (4 or 8 waypoints) are set per task, as Appendix A.5 lists.

Demonstration data. Bare-hand capture records video of the demonstrator’s hands and surroundings; releasing such data requires the demonstrator’s consent and removal of identifying content.