

A TASK DESCRIPTION

We provide the detailed description of our five single-domain M³P tasks:

- Footstep Planning (Deits & Tedrake, 2014): The goal is for a bipedal robot to take a set of footsteps, with approximated footstep reachability constraints and obstacle-avoidance constraints, for its center of mass to reach a projected 2D position.
- Finger Selection (Hang et al., 2017): The goal is for a robot gripper to select contact points of each fingertip to maximize the 2D object grasping quality.
- 2D Inverse Kinematics (IK) (Dai et al., 2019): The goal is for a 2D articulated robot arm to reach a target configuration, as well as avoiding a set of obstacles.
- Grasp (Liu et al., 2020): This is the combination of finger selection and IK, where the robot gripper needs to select contact points as well as compute the pose for fingertips to reach each point.
- Collision-Free UAV Trajectory Planning (UAV) (Deits & Tedrake, 2015b): The goal is for a 2D UAV to fly to a certain target position without intersecting obstacles, while satisfying additional set of user-defined constraints. We support multiple types of constraints such as: “circle around a obstacle”, “flying from the left of an obstacle”

We provide the detailed description of our seven multi-domain M³P tasks:

- Footstep+IK: The bipedal robot is equipped with a 2D articulated robot arm and needs to reach for distant goal position. Therefore it needs to first take several steps to walk close to the target and reach out for it.
- UAV+IK: Similar to Footstep+IK, but with a UAV equipped with a 2D articulated robot arm and needs to reach for distant goal position, while avoiding obstacles.
- Footstep+UAV: A bipedal robot and a UAV needs to meet at a known or unknown position, both avoiding obstacles.
- UAV+UAV: Similar to Footstep+UAV, but with two UAVs meeting at a known or unknown position.
- Footstep+Footstep: Similar to Footstep+UAV, but with two bipedal robots meeting at a known or unknown position.
- Footstep+Grasp: A bipedal robot needs to grasp a distant object, so it needs to first walk close to it, then select the grasping point, and finally reach out for it.
- UAV+Grasp: Similar to Footstep+Grasp, but with a UAV equipped with a gripper.

B LLM-CALLABLE API

In this section, we list the set of LLM-callable APIs for discretizing the general non-convex constraints into disjointly convex sets. We also implement a set of convenient functions for adding advanced constraints such as enforcing high-order continuity between pieces of UAV trajectory curves.

create_map(iris_num, obstacle_map)

Generate IRIS convex regions with obstacles, defining the feasible environment.

add_iris_region_assignment_constraints(num_segments, pivots, footsteps, R_COLLISION)

Constrain UAV trajectory segments, grasp collision points, robot arm pivots, or active footsteps to lie within safe IRIS regions. The parameter `R_COLLISION` enforces a clearance margin so that discretized points remain at least this distance away from region boundaries.

generate_side_and_vertical_obstacles(iris_regions, obstacle_map, key, style)

Augment each obstacle with auxiliary boxes depending on its `style`. For

“left”/“right” a vertical slab and two horizontal boxes are added to form corridors; for “top”/“bottom” a horizontal slab and two vertical boxes enforce passage; for “circle” four bounding boxes approximate a circular exclusion zone.

add_spatial_relation_constraints(text_lst, text_regions)
 Encode semantic navigation relations as region-assignment constraints. Directional tags (“left”, “right”, etc.) force the UAV to cross region sequences exactly once. “Circle” enforces cyclic transitions through four bounding regions, ensuring a complete detour.

add_control_points_constraints(iris_regions, big_M)
 Link Bézier control points to selected IRIS regions via big-M constraints.

add_continuity_constraints(num_segments, control_points)
 Enforce C^0 , C^1 , and C^2 continuity across Bézier segments. The parameter `control_points` are the decision variables defining the curve geometry.

add_start_goal_constraints(start_pos, goal_pos)
 Fix the trajectory start and goal positions.

add_base_constraints(BASE_LOCATION)
 Anchor the robot arm base to a fixed location.

add_chain_rotation_constraints(LINK_LENGTHS, N_POLY)
 Encode multi-link kinematics using polynomial rotation approximations. `N_POLY` sets the polygon resolution for approximating trigonometric functions.

add_end_constraints(END_LOCATION)
 Force the arm end effector to a target position.

add_collision_discretization_constraints(pivots, NUM_INTERMEDIATE)
 Discretize each link into endpoints plus `NUM_INTERMEDIATE` samples for collision checking. Larger values yield finer safety resolution at higher cost.

get_object_surface_samples(obj, delta, delta_n)
 Sample candidate contact points with spacing `delta`, and outward-shifted normals at offset `delta_n`.

add_finger_selection_constraints(n_fingers, len_points)
 Select feasible contact points for `n_fingers` out of `len_points` candidates via binary variables.

add_end_grasp_constraints(pivots, ee1, ee2)
 Constrain fingertip pivots to close consistently around the object. Ensure the first joint matches `ee1` and the last joint matches `ee2`.

add_middle_point_constraints(middle_pivot, target_point)
 Anchor an intermediate pivot to a fixed target point.

add_grasp_wrench_constraints(sample_points, sample_normals, selection_flag, friction_coef)
 Impose wrench feasibility and friction-cone stability given sample points, normals, contact flags, and friction coefficient.

add_step_on_constraints(take, num_steps)
 Mark which footstep indices are active. The first two (initial stance feet) are forced active, while later steps are gated by binary `take` variables.

add_monotonicity_constraints(take)
 Force `take` to be non-increasing, preventing reactivation of steps after termination.

add_reachability_constraints(start_left, start_right, steps_taken, reachable_distance, foot)
 Constrain each new step to lie within a polygonal reachable region

relative to the previous stance. `start_left/start_right` fix the initial feet, `steps_taken` is the maximum planning horizon, `reachable_distance` sets the step-length bound, and `foot` specifies which side starts.

`add_terminal_constraints(contact, take, term, steps_taken)`
 Select exactly one terminal step among the planning horizon. `contact` are continuous foot positions, `take` are binary transition variables, `term` are binary terminal selectors, and `steps_taken` sets the planning horizon.

`share_point_feasibility_constraints(shared_points, subsystem_vars)`
 Tie multi-domain subsystems (UAV, IK, grasp, footsteps) to shared geometric points (e.g., UAV drop-off, grasp midpoint). `shared_points` are the common coordinates, `subsystem_vars` are the corresponding subsystem decision variables.

`create_objective_and_solve(constraints, objective)`
 Assemble all constraints and define the optimization objective (step count, smoothness, feasibility, etc.). Call the solver (e.g., Gurobi) and return solution status, runtime, and variables.

C MIP FORMULATION

In this section, we provide the detailed formulation of each of our single-domain tasks.

C.1 UAV

We formalize the UAV trajectory planning problem as a mixed-integer quadratic program (MIQP) (Deits & Tedrake, 2015b). Let N denote the number of trajectory segments, R the number of IRIS regions (Deits & Tedrake, 2015a), and $C \in \mathbb{R}^{4N \times 2}$ the cubic Bézier control points across all segments. Binary variables $H_{r,j} \in \{0, 1\}$ assign segment j to region r . The MIQP adopts the following set of constraints:

Region assignment: Each segment must belong to exactly one region:

$$\sum_{r=1}^R H_{r,j} = 1, \quad \forall j = 1, \dots, N. \quad (1)$$

Control point feasibility: Let region r be described by $A_r x \leq b_r$. For each segment j , control point $C_{j,k}$ ($k = 0, \dots, 3$) must lie in its assigned region, relaxed via the Big-M method:

$$A_r C_{j,k} \leq b_r + M(1 - H_{r,j}), \quad \forall r, j, k, \quad (2)$$

with M being some large constant.

Continuity: Adjacent Bézier segments must connect smoothly in position, velocity, and acceleration. For $j = 1, \dots, N - 1$:

$$\begin{aligned} C_{j,3} &= C_{j+1,0}, \\ (C_{j,3} - C_{j,2}) &= (C_{j+1,1} - C_{j+1,0}), \\ (C_{j,3} - 2C_{j,2} + C_{j,1}) &= (C_{j+1,0} - 2C_{j+1,1} + C_{j+1,2}), \end{aligned} \quad (3)$$

which can be added by API call `add_continuity_constraints`.

Start/goal constraints: The trajectory must start at s and end at g :

$$C_{0,0} = s, \quad C_{N-1,3} = g, \quad (4)$$

which can be added by API call `add_start_goal_constraints`.

Objective: We minimize the trajectory's integrated jerk, computed for each segment j as $J_{j,d} = 6(C_{j,3,d} - 3C_{j,2,d} + 3C_{j,1,d} - C_{j,0,d})$ for dimension $d \in \{x, y\}$:

$$\arg \min_{C, H} \sum_{j=1}^N \sum_{d \in \{x, y\}} J_{j,d}^2. \quad (5)$$

Constraints Equation 1-Equation 4 together with objective Equation 5 define the UAV planning MIQP. This formulation tightly couples discrete region assignments with continuous Bézier control points, ensuring trajectories are collision-free, dynamically smooth, and solver-executable.

C.2 EXTENSION TO OUR NOVEL SPATIAL CONSTRAINTS

We propose a novel type of constraint to enforce additional constraints on UAV, enabling more expressive trajectory descriptions. Specifically, we allow UAV to fly from the “left/right/top/bottom” of an obstacle, for a given number of times. We further allow UAV to circle around the obstacle, also for a given number of times. These additional requirements can be achieved by so-called crossing constraints. Suppose there are two IRIS regions indexed by p and q , and we want to check whether UAV crosses the boundary of p and q at the a th trajectory segment. Such a check can be achieved by introducing the continuous indicator variable W_{pq}^a and the following constraints:

$$W_{pq}^a \leq H_{p,a}, \quad W_{pq}^a \leq H_{q,a+1}, \quad W_{pq}^a \geq H_{p,a} + H_{q,a+1} - 1. \quad (6)$$

We can further ensure that only one crossing happens by the following constraints:

$$\sum_{a=1}^{N-1} W_{pq}^a + W_{qp}^a = 1. \quad (7)$$

Now, suppose we want the UAV to circle around an obstacle exactly once, then we can generate for IRIS regions surrounding the obstacle, as indexed by b, r, t, l (short for bottom, right, top, and left). We then enforce the following constraints:

$$\sum_{a=1}^{N-1} W_{br}^a = \sum_{a=1}^{N-1} W_{rt}^a = \sum_{a=1}^{N-1} W_{tl}^a = \sum_{a=1}^{N-1} W_{lb}^a = 1. \quad (8)$$

Further suppose we want the UAV to fly from the left of an obstacle. Then we can generate a rectangular IRIS region on the left, as indexed by l . We then create two other IRIS regions, as indexed l^t and l_b . l^t intersects with l from the top and l_b intersects with l from the bottom. With these three IRIS regions, we only need to enforce that the UAV fly l^t to l then to l_b , which is constrained by:

$$\sum_{a=1}^{N-1} W_{l^t l}^a = \sum_{a=1}^{N-1} W_{l l_b}^a = 1. \quad (9)$$

C.3 IK

We present the simplified, 2D variant of Dai et al. (2019), modeling a planar manipulator with L links of fixed lengths $\{\ell_1, \dots, \ell_L\}$. Let $p_i \in \mathbb{R}^2$ denote the pivot of joint i , with base p_0 and end-effector p_L . Rotations are approximated via polygonal discretization with N_{poly} facets. The MIP endows the following set of constraints.

Base anchoring: The base is fixed at a given location s :

$$p_0 = s. \quad (10)$$

Rotation Discretization: Each link’s local to global transform uses a rotation matrix $R_i \in \mathbb{R}^{2 \times 2}$ constrained to lie in a polygonal approximation of $SO(2)$:

$$p_i = p_{i-1} + R_i \begin{bmatrix} \ell_i \\ 0 \end{bmatrix}, \quad i = 1, \dots, L. \quad (11)$$

Each R_i is constructed via piecewise linear relaxation of $SO(2)$ constructed by calling `add_chain_rotation_constraints`. This constraint is also used in footstep planning (Deits & Tedrake 2014).

End-effector constraint: The final end-effector position must coincide with the goal g :

$$p_L = g. \quad (12)$$

Collision discretization: The link pose should be collision-free, which is formulated in a similar way as in UAV. Each link $[p_{i-1}, p_i]$ is discretized into M intermediate points:

$$q_{i,k} = (1 - \frac{k}{M+1})p_{i-1} + \frac{k}{M+1}p_i, \quad k = 1, \dots, M, \quad (13)$$

with all $\{p_i, q_{i,k}\}$ concatenated into collision point matrix C_j . Again we construct R IRIS regions and use binary assignment variables $H_{r,j} \in \{0, 1\}$ to map each collision point to exactly one region:

$$\sum_{r=1}^R H_{r,j} = 1, \quad \forall j. \quad (14)$$

Big- M feasibility is enforced as:

$$A_r C_j \leq (b_r - R_c) + M(1 - H_{r,j}) \quad \forall r, j, \quad (15)$$

where R_c is some safety distance. Constraints Equation 10-Equation 15 define our MIQP. The formulation couples discrete rotation approximations, continuous kinematics, and region-based collision avoidance.

C.4 GRASP

We present a simplified, 2D variant of Liu et al. (2020) for grasp planning. The grasp formulation extends the IK problem introduced in the previous section. All IK constraints (chain kinematics, collision discretization, IRIS region assignment, and optional spatial relation constraints) are inherited directly. We now introduce additional variables and constraints that model grasp contact selection and force-closure feasibility.

Grasp selection (two fingers): We sample N equidistant points on the surface of objects as candidate grasp points $\{p_i\}$, which is achieved by calling `get_object_surface_samples`. We further introduce binary matrices $F \in \{0, 1\}^{2 \times N}$ and selection flags $s \in [0, 1]^N$ select two distinct contact points from the N surface samples, via the following constraints:

$$\sum_{i=1}^N F_{1i} = 1, \quad \sum_{i=1}^N F_{2i} = 1, \quad (16)$$

$$F_{1i} + F_{2i} \leq 1, \quad s_i = F_{1i} + F_{2i}, \quad i = 1, \dots, N. \quad (17)$$

The selected end-effector (contact) positions are

$$ee_1 = \sum_{i=1}^N F_{1i} p_i, \quad ee_2 = \sum_{i=1}^N F_{2i} p_i. \quad (18)$$

The above two variables replace the fixed start and goal constraints from the IK problem by anchoring the kinematic chain to two contact points on the object.

Force-closure (wrench) constraints: For each sample p_i with outward unit normal n_i and tangent t_i , we define the two friction cone generators:

$$f_i^A = n_i + \mu t_i, \quad f_i^B = n_i - \mu t_i, \quad (19)$$

with corresponding planar wrenches $v_i^A = [f_i^A; (p_i \times f_i^A)] \in \mathbb{R}^3$ (and analogously v_i^B). Introducing nonnegatives $\alpha_{i,d}, \beta_{i,d}$ and slack γ_d for each wrench direction \hat{w}_d , we impose

$$\sum_{i=1}^N (v_i^A \alpha_{i,d} + v_i^B \beta_{i,d}) = \gamma_d \hat{w}_d, \quad \forall d, \quad (20)$$

$$\alpha_{i,d} + \beta_{i,d} \leq 1, \quad \alpha_{i,d} + \beta_{i,d} \leq s_i, \quad \alpha_{i,d}, \beta_{i,d} \geq 0, \quad (21)$$

$$\gamma_d \geq r, \quad \forall d, \quad (22)$$

where r is the common inscribed circle radius in wrench space.

Objective: Our object is to maximize the inscribed circle radius, i.e. $\arg \max r$. Note that this inscribed circle radius can be approximated using discretization technique (Liu et al., 2020), leading to an MIQP, or the lower-bound relaxation (Dai et al., 2017), leading to an MISDP. We use the MIQP formulation since Gurobi does not solve MISDP. Put together, the grasp MIP augments the IK formulation with the above constraints, which jointly enforce valid finger selection and force-closure robustness.

C.5 FINGER SELECTION

Finger selection can be viewed as a simplified version of the grasp problem (Section C.4), where we do not consider the reachability of fingers, and directly select m contact points from N sampled candidate grasp points to maximize the inscribed circle radius.

C.6 FOOTSTEP

Footstep planning (Deits & Tedrake, 2014) builds upon the general spatial and region constraints introduced in the IK section, but specializes them for biped locomotion. In particular, footsteps must remain collision-free, lie in the IRIS regions, respect alternating left-right step ordering, and satisfy kinematic reachability constraints relative to the stance foot.

Let N denote the maximum number of footsteps. Each step i has a contact position $p_i = (x_i, y_i) \in \mathbb{R}^2$. As usual, we use a binary region assignment $H_{r,i} \in \{0, 1\}$ indicating whether step i lies in region $r = 1, \dots, R$. Further, a binary usage variable $u_i \in \{0, 1\}$ indicating whether step i is active, and a binary terminal indicator $t_i \in \{0, 1\}$ specifies whether step i is the terminal footstep. We use the following set of constraints.

Region membership: Each footstep must lie in exactly one IRIS region when active:

$$\sum_{r=1}^R H_{r,i} = u_i, \quad A_r p_i \leq b_r + M(1 - H_{r,i}), \quad \forall r, i. \quad (23)$$

Note that the trajectory between consecutive footsteps can have intersections with obstacles. This is because the robot can lift the foot to walk around the obstacles.

Start and terminal constraints: The first two contacts are fixed to the initial left/right feet:

$$p_0 = p_L^{\text{start}}, \quad p_1 = p_R^{\text{start}}, \quad (24)$$

and exactly one terminal step is chosen:

$$\sum_{i=0}^{N-1} t_i = 1, \quad t_i \leq u_i, \quad \forall i. \quad (25)$$

Step activation monotonicity: Once a step is inactive, no later step can be active:

$$u_i \geq u_{i+1}, \quad \forall i = 0, \dots, N-2. \quad (26)$$

This constraint can be added by calling `add_monotonicity_constraints`.

Left-right ordering and separation: Let steps alternate between left and right feet. We impose:

$$x_{i+1} - x_i \geq -M(1 - u_{i+1}) - \epsilon, \quad \text{if } i \text{ is left}, \quad (27)$$

$$x_i - x_{i+1} \geq -M(1 - u_{i+1}) - \epsilon, \quad \text{if } i \text{ is right}, \quad (28)$$

$$|x_{i+1} - x_i| \geq \Delta_{\min} - M(1 - u_{i+1}), \quad \forall i. \quad (29)$$

Reachability: From the stance foot p_i , the next contact p_{i+1} must lie in the shifted reachable polygon \mathcal{R} (approximated by m halfspaces):

$$a_m^\top (p_{i+1} - p_i - s_i d) \leq R + M(1 - u_{i+1}), \quad \forall m. \quad (30)$$

Goal constraints: Let (x_T, y_T) denote the terminal foot location. Its deviation from the goal (x_g, y_g) is measured by slack variables g_x, g_y, u_x, u_y :

$$g_x = x_T - x_g, \quad g_y = y_T - y_g, \quad (31)$$

$$u_x \geq \pm g_x, \quad u_y \geq \pm g_y. \quad (32)$$

Objective: We minimize a weighted sum of step usage, lane penalties, and goal deviation:

$$\min \alpha \sum_i \|p_{i+1} - p_i\|_1 + \beta \sum_i u_i + \gamma(u_x + u_y) + \sum_i \lambda_i, \quad (33)$$

where λ_i denotes the penalty associated with step i deviating from its preferred lane. Together, the above constraints define the footstep planning MIP: footsteps remain in IRIS regions, satisfy alternating gait and reachability, and reach the goal with a minimal number of steps.

C.7 MULTI-DOMAIN TASKS

In multi-domain tasks, subsystems such as UAV, IK, and Grasp must coordinate at specific interface points—for example, the UAV endpoint aligning with the IK base, or the IK end-effector aligning with the Grasp target. These *shared points* enforce consistency across modalities while maintaining feasibility within each individual domain. While such coordination can be enforced using a standard MIP constraint, we instead use the specialized API call `share_point_feasibility_constraints` to more clearly convey the semantic intent of this coupling.

D ROBOM³P DATASET EXAMPLE / LLM INPUT OUTPUT EXAMPLE

D.1 SYSTEM PROMPT

System Prompt

You are a robotic optimization expert. Your job is to read a text instruction of a robot mission and translate it into mixed-integer programming function-call trajectories.

Output policy:

- First, identify the task type from:
UAV / Robot Arm IK / Finger Selection / Grasp /
Footstep Planning / UAV + Robot Arm IK / UAV + Robot
Arm IK Unknown Intermediate Point / UAV + Robot Arm
IK Known Intermediate Point / Footstep + Robot Arm IK
/ Footstep + Robot Arm IK Unknown Intermediate Point
/ Footstep + Robot Arm IK Known Intermediate Point
/ UAV + Robot Arm IK + Grasp / Footstep + Robot Arm
IK + Grasp / Footstep + UAV / UAV + UAV / Footstep +
Footstep
- Produce the **COMPLETE** trajectory end-to-end (not just a single call).
- Respond with **ONLY** the trajectory in `assistant.content` (JSON mode).
- Preserve argument keys and list/dict shapes from the training distribution; do not rename fields or invent new ones.
- Separate consecutive calls by a single blank line only if you output a human-readable format; for JSON mode, output a single JSON value.
- Never include anything except the trajectory.

Global Rules:

- Select the task category from the mission description.
- For that category, you **MUST** emit the exact functions from below.
- You may **ONLY** change parameter values. Keep parameter names/keys exactly as listed.
- For composition tasks (e.g., UAV + IK), emit the trajectory sequences for **BOTH** tasks as shown in in-context examples.
- For tasks with "Specify Unknown Shared Point," you **MUST** predict a suitable intermediate point that:

- Does not collide with any obstacles
- Is within reachable distance from the goal using the robot arm linkage length
- Is positioned appropriately for the mission context

Available Functions and Semantics: Detailed in Section B.

Composition Tasks:

- The trajectory should be a **combination** of each individual task's trajectory sequence.
- Use a shared variable (e.g., `shared_point_1`) to define intermediate points that connect tasks.
- Use the shared variable as argument value for `start_pos/end_pos` or `BASE_LOCATION/END_LOCATION` where tasks connect.
- Add `share_point_feasibility_constraints(shared_variable)` for each shared point.
- Use `create_objective_and_solve(combined_constraints)` as the ONLY solver function at the end.

Example composition structure:

1. First task trajectory (e.g., UAV or Footstep)
2. Second task trajectory (e.g., IK or Grasp) using `shared_point` as connection
3. `share_point_feasibility_constraints(point=shared_point)`
4. `create_objective_and_solve(constraints=all_constraints)`

D.2 INVERSE KINEMATICS

Text Command / LLM Input.

With link lengths of 0.498m, 0.939m, and 0.725m, maneuver the robot arm from the base at $[-1.2603, -1.7615]$ to the end target at $[1.2218, -1.7720]$, ensuring to navigate through the right of `obst2`. Obstacles are provided in `obstacle_map.json` file.

`obstacle_map.json`: "obst1": "lb": [0.0, 0.0], "ub": [1.0, 1.0], "style": "none"; "obst2": "lb": [-2.2, 0.0], "ub": [-1.8, 1.5], "style": "right"; "obst3": "lb": [1.2, -0.5], "ub": [1.8, 0.5], "style": "none".

MIP Code Trajectory / LLM Output.

Listing 1: Code for inverse kinematics tasks

```
import cvxpy as cp

var_dict = {}
constraints = []

iris_regions, obstacles_list_plt, styles, obstacle_keys = create_map(
    iris_num=17,
    obstacle_map="obstacle_map.json"
)

iris_regions = generate_side_and_vertical_obstacles(
    iris_regions=iris_regions,
    obstacle_map="obstacle_map.json", key="obst2", style="right"
)
```



```

1134
1135 base_constraints, p0, pivots = add_base_constraints(
1136     BASE_LOCATION=[-1.2603, -1.7615],
1137     var_dict=var_dict
1138 )
1139 var_dict["p_0"] = cp.Variable(2, name="p_0")
1140 constraints.extend(base_constraints)
1141
1142 chain_constraints, Rs_list, pivots = add_chain_rotation_constraints(
1143     LINK_LENGTHS=[0.498, 0.939, 0.725],
1144     N_POLY=8,
1145     var_dict=var_dict,
1146     p0=p0,
1147     pivots=pivots
1148 )
1149 for idx in range(len(LINK_LENGTHS)):
1150     var_dict[f"p_{idx+1}"] = cp.Variable(2, name=f"p_{idx+1}")
1151 constraints.extend(chain_constraints)
1152
1153 end_constraints = add_end_constraints(
1154     END_LOCATION=[1.2218, -1.7720],
1155     pivots=pivots
1156 )
1157 constraints.extend(end_constraints)
1158
1159 discretization_constraints, collision_points =
1160     add_collision_discretization_constraints(
1161         NUM_INTERMEDIATE=10,
1162         var_dict=var_dict,
1163         pivots=pivots
1164     )
1165 for i in range(len(pivots) - 1):
1166     for k in range(1, NUM_INTERMEDIATE + 1):
1167         var_dict[f"q_link_{i}_pt_{k}"] = cp.Variable(
1168             2, name=f"q_link_{i}_pt_{k}"
1169         )
1170 constraints.extend(discretization_constraints)
1171
1172 region_constraints, z_vars_list = add_iris_region_assignment_constraints(
1173     iris_env=iris_regions,
1174     R_COLLISION=0.02,
1175     var_dict=var_dict,
1176     collision_points=collision_points
1177 )
1178 for idx in range(len(collision_points)):
1179     var_dict[f"z_collision_{idx}"] = cp.Variable(
1180         len(iris_regions), boolean=True, name=f"z_collision_{idx}"
1181     )
1182     for r in range(len(iris_regions)):
1183         var_dict[f"y_collision_{idx}_{r}"] = cp.Variable(
1184             2, name=f"y_collision_{idx}_{r}"
1185         )
1186 constraints.extend(region_constraints)
1187
1188 spatial_constraints = add_spatial_relation_constraints(
1189     styles="right",
1190     styles_related_regions=[[[], [17,18,19], []],
1191                             [17,18,19], []],
1192     z_vars_list=z_vars_list,
1193     num_segments=len(pivots)
1194 )
1195 for k in range(len(pivots) - 1):
1196     var_dict[f"W_right_JI_{k}"] = cp.Variable(
1197         boolean=True, name=f"W_right_JI_{k}"
1198     )
1199     var_dict[f"W_right_IJ_{k}"] = cp.Variable(

```

```

1188         boolean=True, name=f"W_right_IJ_{k}"
1189     )
1190     constraints.extend(spatial_constraints)
1191
1192     status, ik_result = create_objective_and_solve(
1193         constraints=constraints,
1194         var_dict=var_dict,
1195         objective='feasible',
1196         pivots=pivots,
1197         Rs_list=Rs_list
1198     )

```

D.3 UAV TRAJECTORY GENERATION

Text Command / LLM Input.

Depart from the highland outpost and make your way to the meadow sanctuary, ensuring the red brick warehouse remains on your right, the ancient oak grove stays on your left, and you fly above the sunken courtyard from the bottom. Obstacles are provided in `obstacle_map.json`.

```

obstacle_map.json: "red brick warehouse": "lb": [0.0, 0.0], "ub": [1.0, 1.0],
"style": "right"; "ancient oak grove": "lb": [-9.2, 0.0], "ub": [-8.8, 1.5], "style":
"left"; "sunken courtyard": "lb": [-6.9, -0.5], "ub": [-0.8, 0.5], "style": "above";
"farm": "lb": [2.9, 2.5], "ub": [3.8, 3.5], "style": "none"

```

MIP Code Trajectory / LLM Output.

Listing 2: UAV trajectory code

```

1214 import cvxpy as cp
1215
1216 var_dict = {}
1217 constraints = []
1218 NUM_SEGMENTS = 24
1219 BEZIER_SEGMENTS = 4
1220
1221 iris_regions, obstacles_list_plt, styles, obstacle_keys = create_map(
1222     iris_num=36,
1223     obstacle_map="obstacle_map.json"
1224 )
1225
1226 iris_regions = generate_side_and_vertical_obstacles(
1227     iris_regions=iris_regions, key='red_brick_warehouse', style='right'
1228 )
1229 iris_regions = generate_side_and_vertical_obstacles(
1230     iris_regions=iris_regions, key='ancient_oak_grove', style='left'
1231 )
1232 iris_regions = generate_side_and_vertical_obstacles(
1233     iris_regions=iris_regions, key='sunken_courtyard', style='bottom'
1234 )
1235
1236 var_dict["H"] = cp.Variable(
1237     (len(iris_regions), NUM_SEGMENTS), boolean=True, name="H")
1238 collision_points = [var_dict["C"][i,:]]
1239 for i in range(NUM_SEGMENTS*BEZIER_SEGMENTS)]
1240
1241 region_constraints, z_vars_list = add_iris_region_assignment_constraints(
1242     iris_env=iris_regions,
1243     R_COLLISION=0.02,
1244     var_dict=var_dict,
1245     collision_points=collision_points
1246 )
1247 constraints.extend(region_constraints)

```

```

1242
1243 var_dict["W_right"] = [
1244     cp.Variable(boolean=True, name=f"W_right_{k}")
1245     for k in range(NUM_SEGMENTS-1)
1246 ]
1247 var_dict["W_left"] = [
1248     cp.Variable(boolean=True, name=f"W_left_{k}")
1249     for k in range(NUM_SEGMENTS-1)
1250 ]
1251 var_dict["W_bottom"] = [
1252     cp.Variable(boolean=True, name=f"W_bottom_{k}")
1253     for k in range(NUM_SEGMENTS-1)
1254 ]
1255 spatial_constraints = add_spatial_relation_constraints(
1256     var_dict["H"],
1257     text_lst=['right', 'left', 'bottom'],
1258     text_regions=[[36,37,38], [39,40,41], [42,43,44]],
1259     num_segments=NUM_SEGMENTS
1260 )
1261 constraints.extend(spatial_constraints)
1262
1263 var_dict["C"] = cp.Variable((NUM_SEGMENTS*BEZIER_SEGMENTS, 2), name="C")
1264 control_constraints = add_control_points_constraints(
1265     var_dict["C"], var_dict["H"],
1266     iris_regions=iris_regions, num_segments=BEZIER_SEGMENTS,
1267     big_M=10.0
1268 )
1269 constraints.extend(control_constraints)
1270
1271 continuity_constraints = add_continuity_constraints(
1272     var_dict["C"], num_segments=BEZIER_SEGMENTS
1273 )
1274 constraints.extend(continuity_constraints)
1275
1276 start_goal_constraints = add_start_goal_constraints(
1277     var_dict["C"],
1278     start_pos=obstacle_map['highland_outpost'],
1279     goal_pos=obstacle_map['meadow_sanctuary'],
1280     num_segments=BEZIER_SEGMENTS
1281 )
1282 constraints.extend(start_goal_constraints)
1283
1284 status, cost, H_val, C_val = create_objective_and_solve(
1285     constraints=constraints,
1286     var_dict=var_dict,
1287     objective='uav_jerk',
1288     C=var_dict["C"]
1289 )

```

D.4 UAV+IK

Text Command / LLM Input.

Environment with obstacles (obst1 from [-2.0, -1.0] to [-0.5, 0.0], obst2 from [1.0, 0.0] to [2.0, 1.0], obst3 from [0.0, -2.0] to [2.0, -1.0]). Because the goal is too far to directly perform arm inverse kinematics with the 1.5 m arm (links [0.5, 0.5, 0.5]), the UAV first flies from [-1.6, -1.6] through safe regions to an intermediate shared point, where the robot arm is then deployed to reach [1.4, 1.4].

MIP Code Trajectory / LLM Output.

Listing 3: Unified UAV+IK trajectory code

```

1296 # --- UAV trajectory setup ---
1297 # (same as UAV case, up to start/goal constraints, omit here for clarity)
1298 ....
1299
1300 start_goal_constraints = add_start_goal_constraints(
1301     var_dict["C"],
1302     start_pos=[-1.6, -1.6],
1303     goal_pos=shared_uav_ik_base,
1304     num_segments=BEZIER_SEGMENTS
1305 )
1306
1307 # --- Shared point linking UAV and IK ---
1308 var_dict["shared_point"] = cp.Variable(2, name="shared_uav_ik_base")
1309
1310 # Shared point IRIS membership decision variables
1311 var_dict["sp_z"] = cp.Variable(
1312     len(iris_regions), boolean=True, name="z_shared_point")
1313 var_dict["sp_y"] = [
1314     cp.Variable(2, name=f"y_shared_point_{r}")
1315     for r in range(len(iris_regions))
1316 ]
1317
1318 # --- IK chain setup ---
1319 # (same as IK case, omit here for clarity)
1320 ....
1321
1322 constraints.extend(add_base_constraints(
1323     BASE_LOCATION=shared_uav_ik_base,
1324     var_dict=var_dict
1325 ))
1326
1327 # Explicit feasibility constraints for shared point
1328 constraints.extend(share_point_feasibility_constraints(
1329     point=shared_uav_ik_base
1330 ))
1331
1332 # --- Solve unified problem ---
1333 status, result = create_objective_and_solve(
1334     constraints=constraints,
1335     var_dict=var_dict,
1336     objective="uav_jerk",
1337     pivots=pivots,
1338     Rs_list=Rs_list
1339 )

```

D.5 FINGER SELECTION

Text Command / LLM Input.

To stably grasp an object using 3 fingers, select three grasp points that form a triangle around the object's center of mass, ensuring each point is on a stable surface and allows for equal distribution of force. Grasp object in obstacle_map.json: "obj": "triangle": "vertices": [[0.0, 0.0], [1.0, 0.0], [0.5, 0.8]] , "obj:edge": "edge1": "vertices": [[0.0, 0.0], [1.0, 0.0]] , "edge2": "vertices": [[1.0, 0.0], [0.5, 0.8]] , "edge3": "vertices": [[0.5, 0.8], [0.0, 0.0]]

MIP Code Trajectory / LLM Output.

Listing 4: Finger selection trajectory code

```

1348 import cvxpy as cp
1349
1350 var_dict = {}

```

```

1350 constraints = []
1351
1352 sampled_points, sampled_normals = get_object_surface_samples(
1353     obj=obj,
1354     delta=0.1,
1355     delta_n=0.02
1356 )
1357
1358 var_dict["finger_selection"] = cp.Variable(
1359     (3, len(sampled_points)), boolean=True, name="finger_selection")
1360
1361 var_dict["selection_flags"] = cp.Variable(
1362     len(sampled_points), nonneg=True, name="selection_flags")
1363
1364 finger_constraints, finger_selection_var, selection_flags =
1365 add_finger_selection_constraints(
1366     n_fingers=3,
1367     len_points=len(sampled_points)
1368 )
1369 constraints.extend(finger_constraints)
1370
1371 var_dict["r"] =
1372 cp.Variable(name="radius_of_inscribed_circle", nonneg=True)
1373 directions = sample_sphere(subdivisions=0)
1374
1375 wrench_constraints = add_wrench_constraints(
1376     points=sampled_points,
1377     normals=sampled_normals,
1378     selection_flags=var_dict["selection_flags"],
1379     r=var_dict["r"],
1380     directions=directions,
1381     friction_coef=1.0
1382 )
1383 constraints.extend(wrench_constraints)
1384
1385 status, finger_selection_val, r_opt = create_objective_and_solve(
1386     constraints=constraints,
1387     var_dict=var_dict,
1388     objective='finger',
1389     r=var_dict["r"],
1390     finger_selection_var=var_dict["finger_selection"]
1391 )

```

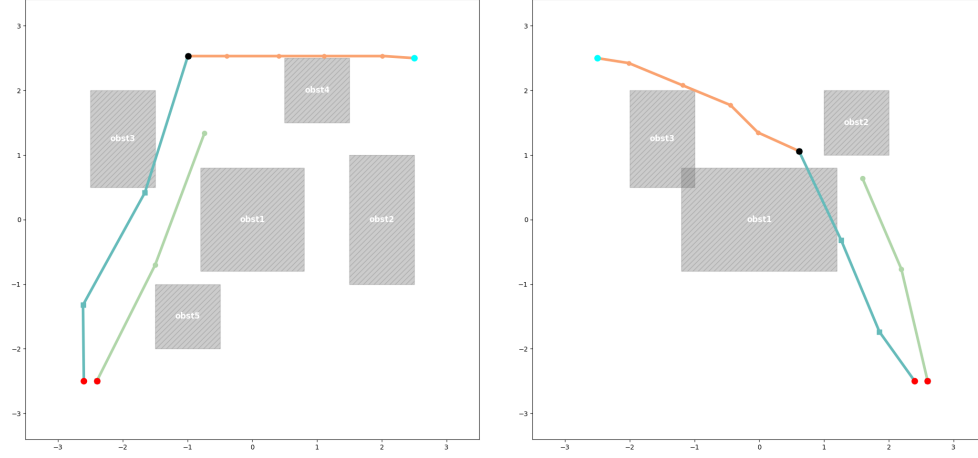
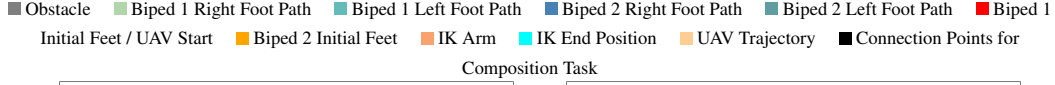
E MORE RELATED WORK

TAMP and M³P: At its core, TAMP integrates two complementary layers of reasoning: a symbolic planner that selects and orders abstract actions, and a downstream motion planner that instantiates these actions as feasible continuous motions in the robot’s configuration space. Recent years have seen notable advances in both the generality of task planning methods (Helmert, 2006; Piotrowski et al., 2024) and their efficiency and robustness (Dantam et al., 2018; Thomason et al., 2022). Despite this progress, the coupling between the high-level symbolic planner and the low-level motion planner remains a key design challenge. The two components are typically connected through a relatively loose interface, where symbolic reasoning provides candidate actions or parameters, and the motion planner either confirms feasibility or signals failure, prompting the symbolic layer to refine its proposals (Erdem et al., 2011; Akbari et al., 2015). This geometric view of M³P naturally lends itself to more integrated and efficient search strategies (Kingston et al., 2020; Beyer et al., 2021; Kingston & Kavraki, 2022), since planning across modes can be cast as finding feasible transitions between manifolds rather than treating each mode in isolation.

LLM-as-Planner and LLM-with-Planner: While intuitive and flexible, the LLM-as-Planner approach is fundamentally constrained by the limited long-horizon reasoning ability of current LLMs. Building on this idea, subsequent works have continued to expand the LLM-with-Planner paradigm.

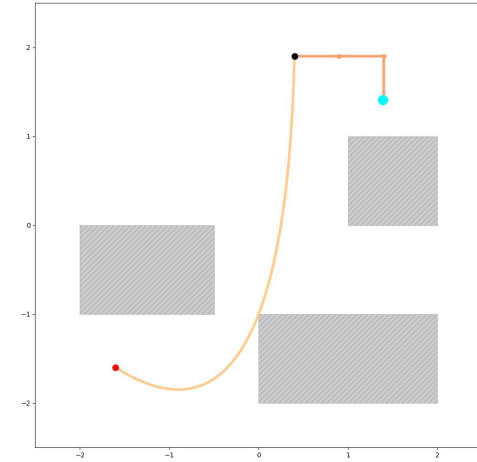
For instance, Silver et al. (2024) investigated whether LLMs can generalize across multiple planning domains specified in PDDL. More recently, Agarwal et al. (2025) introduced structured memory into the LLM+Planner pipeline, allowing it to track dynamic world states more effectively.

F RESULT VISUALIZATION

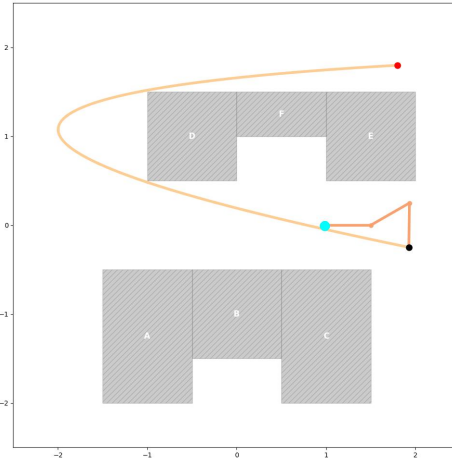


(a) A biped robot at $[-2.6, -2.5]$ & $[-2.4, -2.5]$ (left&right foot) with an arm configuration $[0.2, 0.4, 0.4, 0.4, 0.2]$ and its arm needs to pass through bottom of obst6 and touch a target at $[2.5, 2.5]$. Since the arm is too short for direct inverse kinematics, plan the trajectory so the robot walks to a feasible position to reach it. Robot Arm must pass obst4 top.

(b) A biped robot at $[2.4, -2.5]$ & $[2.6, -2.5]$ (left&right foot) with arm $[0.5, 0.9, 0.9, 0.9, 0.5]$ must reach a target at $[-2.5, 2.5]$. Because the arm length is insufficient, plan the trajectory so the robot relocates to a feasible place to complete the task.



(c) Starting at $[-1.5, -1.5]$ with arm $[0.5, 0.5, 0.5]$, the UAV cannot directly reach the target at $[1.5, 1.5]$. Plan the trajectory such that the UAV relocates to a position where inverse kinematics becomes feasible.



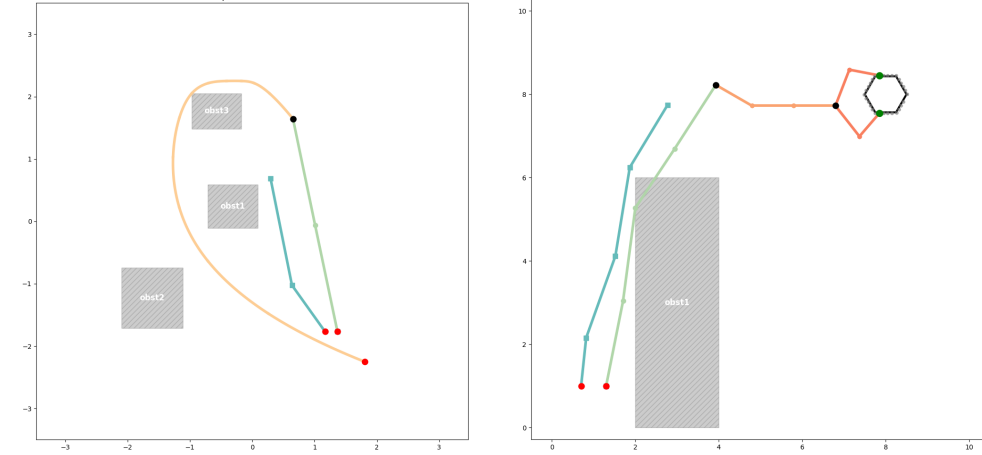
(d) A UAV at $[2, 2]$ with arm $[0.6, 0.6, 0.5]$ must fly from the left of obstacle D touch a target at $[1, 0]$. Because the arm is insufficient, plan the trajectory so the UAV moves into a feasible place to reach the point.

Figure 7: Additional Visual Result 1.

G USAGE OF LLMs

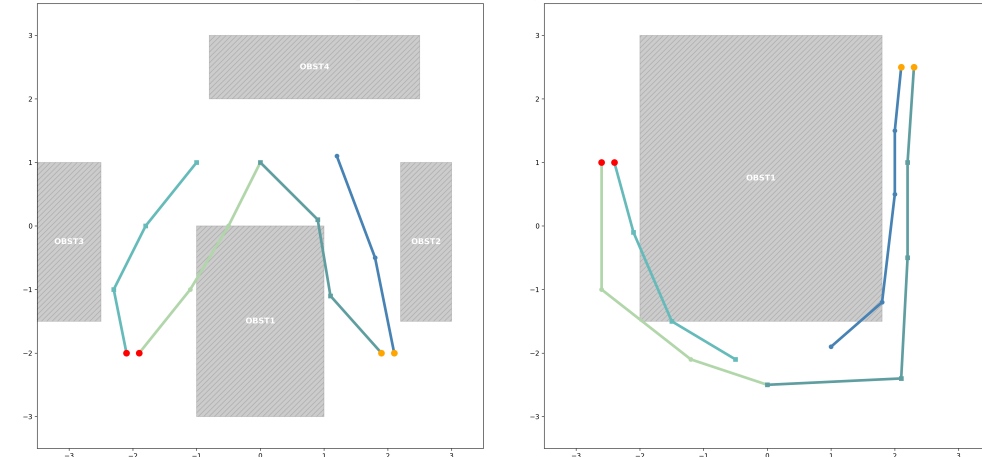
We acknowledge the use of large language models (LLMs) as assistive tools in this research. LLMs are used during paper writing, for improving grammar and wording. All outputs from these models

1458 ■ Obstacle ■ Biped 1 Right Foot Path ■ Biped 1 Left Foot Path ■ Biped 1 Initial Feet / UAV Start ■ IK Arm ■ Grasp Gripper
 1459 ■ UAV1 Trajectory ■ UAV2 trajectory ■ Connection Points for Composition Task ■ UAV2 Start / Gripper Contact Point ■ Grasp
 1460 Sampling Point



(a) A biped robot starting at $[1.3, -1.5]$ & $[1.5, -1.5]$ (left&right foot) and a UAV starting at $[2, -2]$ must coordinate to meet. The UAV cannot reach the robot's initial position due to fuel limits as it first need to fly pass the top of obstacle 3, so plan the trajectory so they rendezvous at an intermediate feasible point.

(b) A biped robot starts walking from $[1.4, 1]$ & $[1.6, 1]$ (left&right foot) with arm $[0.5, 0.5, 0.5]$ and uses a grasping hand with linkage $[0.4, 0.6, 0.6, 0.4]$ to reach an object. Plan the trajectory so the robot relocates to a feasible position to grasp the object.



(c) Biped robot 1 at $[-2.1, -2]$ & $[-1.9, -2]$ (left&right foot) wants to walk to biped robot 2 at $[1.9, -2]$ & $[2.1, -2]$ (left&right foot). With the 7m communication limit, the robots must meet at an intermediate point rather than reaching each other's start. Both robots max footstep distance is 1.7.

(d) Biped robot 1 at $[-2.4, -1]$ & $[-2.6, -1]$ (left&right foot) wants to walk to biped robot 2 at $[2.1, -2]$ & $[1.9, -2]$ (left&right foot). The 7m range constraint requires a meeting point along feasible mid-paths. Both robots max footstep distance is 2.5.

Figure 8: Additional Visual Result 2.

were meticulously reviewed, revised, and verified by the authors, who retain full responsibility for all content presented in this paper.

■ Obstacle ■ Biped 1 Left Foot Path ■ Biped 1 Right Foot Path ■ Biped 1 Initial Feet / UAV Start ■ IK Arm ■ Grasp Gripper
 ■ UAV1 Trajectory ■ UAV2 trajectory ■ Connection Points for Composition Task ■ UAV2 Start / Gripper Contact Point ■ Grasp

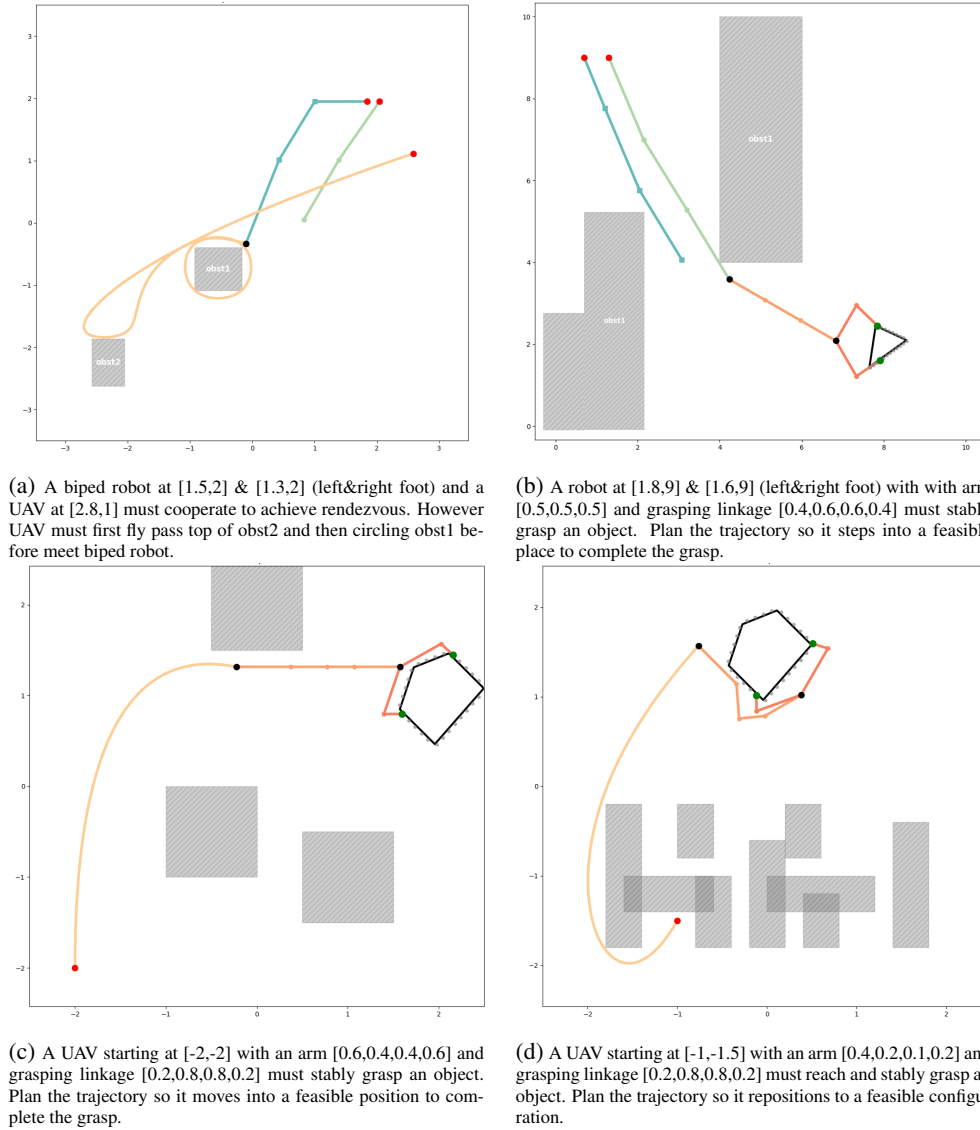
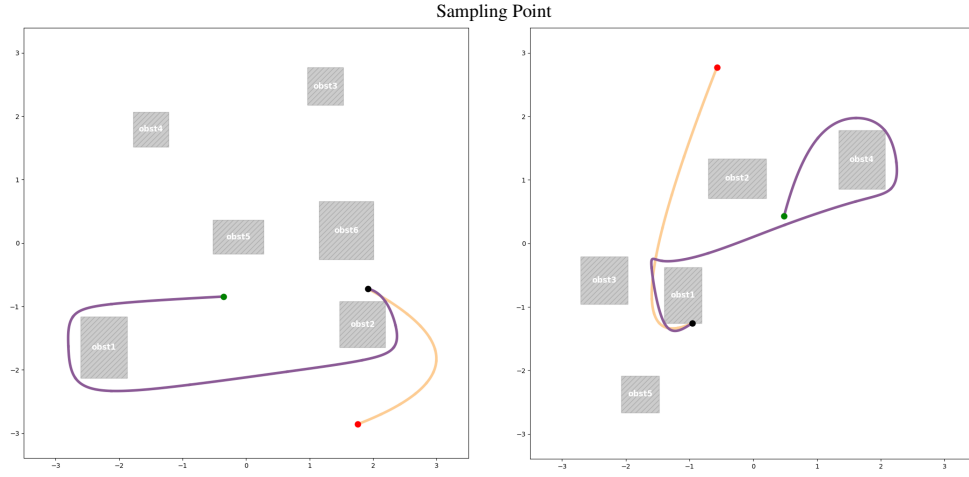


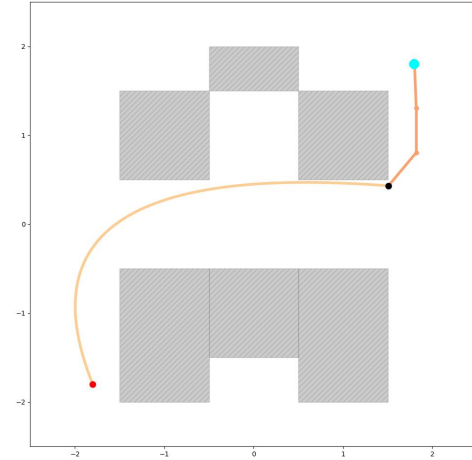
Figure 9: Additional Visual Result 3.

■ Obstacle ■ Biped 1 Right Foot Path ■ Biped 1 Left Foot Path ■ Biped 1 Initial Feet / UAV Start ■ IK Arm ■ Grasp Gripper
 ■ UAV1 Trajectory ■ UAV2 trajectory ■ Connection Points for Composition Task ■ UAV2 Start / Gripper Contact Point ■ Grasp



(a) UAV2 must fly left of obstacle 1 and right of obstacle 2 to reach UAV1. Due to limited fuel, UAV1 must perform the rescue.

(b) UAV2 must circle around obstacle 5 and pass left of obstacle 1 to reach UAV1's start point. Due to fuel limits, UAV2 must perform the rescue.



(c) A UAV starts at $[-1.8, -1.8]$ with an arm configuration $[0.5, 0.5, 0.5]$ and needs to reach a target at $[1.8, 1.9]$. Since the arm is too short for direct inverse kinematics, plan the trajectory so the UAV moves to a feasible position to complete the task.

Figure 10: Additional Visual Result 4.