Supplementary Material
Collaborative Planning with Concurrent Synchronization for Operationally Constrained UAV-UGV Teams

Author Names Omitted for Anonymous Review. Paper-ID 802.

I. Mixed-Integer Programming for Groundtruth Generation

We adopt an imitation learning framework to train CoPCS, where the ground-truth demonstrations are obtained by solving a Mixed-Integer Programming (MIP) problem. The formulation is defined as follows:

$$\min \sum_{\substack{i,j\in\mathcal{T}^m\cup\mathcal{T}^p \\ k\in|\mathcal{U}^a|}} d(\mathbf{d}_i,\mathbf{d}_j)\cdot x_{i,j,k} + \sum_{\substack{i,j\in\mathcal{T}^p \\ l\in|\mathcal{U}^g|}} d(\mathbf{d}_i^p,\mathbf{d}_j^p)\cdot z_{i,j,l} \tag{1}$$

$$\text{s.t.} \sum_{k\in|\mathcal{U}^a|} y_{i,k} = 1, \quad \forall i\in\mathcal{T}^m \tag{2}$$

$$\sum_j x_{i,j,k} = y_{i,k}, \quad \forall i,\forall k \tag{3}$$

$$\sum_j x_{j,i,k} = y_{i,k}, \quad \forall i,\forall k \tag{4}$$

$$x_{i,i,k} = 0, \quad \forall i,\forall k \tag{5}$$

$$0 \le f_{i,k} \le f_{\text{capacity}}, \quad \forall i,\forall k \tag{6}$$

$$f_{j,k} \le f_{i,k} - c(\mathbf{d}_i,\mathbf{d}_j)\cdot x_{i,j,k} + M(1-x_{i,j,k}), \quad \forall i,j,\forall k \tag{7}$$

$$f_{i,k} \ge f_{\text{capacity}}\cdot \sum_{l,t} w_{k,l,t}, \quad \forall i,\forall k \tag{8}$$

$$u_{i,k} - u_{j,k} + n\cdot x_{i,j,k} \le n-1, \quad \forall i,j\in\mathcal{T}^m, i\ne j,\forall k \tag{9}$$

$$\sum_j z_{i,j,l} = \sum_j z_{j,i,l}, \quad \forall i\in\mathcal{T}^p,\forall l \tag{10}$$

$$z_{i,i,l} = 0, \quad \forall i\in\mathcal{T}^p,\forall l \tag{11}$$

$$w_{k,l,t} \le \sum_j z_{j,i,l}, \quad \forall k,l,t,\forall i \tag{12}$$

$$\sum_k w_{k,l,t} \le 1, \quad \forall l,t \tag{13}$$

where the binary variable $x_{i,j,k}\in\{0,1\}$ indicates whether UAV k travels from node i to node j, and $y_{i,k}\in\{0,1\}$ indicates whether UAV k visits task i. Similarly, $z_{i,j,l}\in\{0,1\}$ represents whether UGV l moves from path point i to path point j, while $w_{k,l,t}\in\{0,1\}$ specifies whether UAV k recharges at UGV l at time t. To eliminate subtours, we introduce an integer variable $u_{i,k}\in\{0,1,\ldots,n-1\}$ for UAV k at node i. Finally, $f_{i,k}\ge 0$ denotes the continuous variable representing the energy level of UAV k at node i.

The objective function is defined in Eq. 1, minimizing the total travel distance for UAVs and the total travel distance for UGVs. Eq. 2 ensures that each task point is visited exactly once by any UAV. Eq. 3 ensures that if UAV k visits node i, it must have exactly one outgoing edge from that node. Eq. 4 ensures that if UAV k visits node i, it must have exactly one incoming edge to that node. Eq. 5 ensures UAVs cannot travel from a node to itself. Eq. 6 ensures UAV energy levels remain between empty and full capacity. Eq. 7 ensures UAV energy decreases by the consumption amount when traveling between two nodes, using big-M to deactivate the constraint when no travel occurs. Eq. 8 ensures UAV energy levels become full capacity when recharging occurs. Eq. 9 uses Miller-Tucker-Zemlin subtour elimination to prevent disconnected cycles in UAV paths. Eq. 10 ensures UGV flow conservation where incoming edges equal outgoing edges at each path point. Eq. 11 ensures UGVs cannot move from a path point to itself. Eq. 12 ensures that when UAV k recharges at UGV l, the UGV must be positioned at the recharge location. Eq. 13 ensures that each UGV can serve at most one UAV for recharging at any given time.

II. Implementation of CoPCS

To implement our CoPCS, we construct the edges of the graph by connecting nearby entities within a 5 km radius. The graph transformer network ψ first projects the attributes of each node $\mathbf{t}_i$ into a unified feature space. The projection matrices $\mathbf{W}^z$ are

configured with dimensions 3×128 for tasks, 2×128 for paths, 4×128 for UAVs, and 4×128 for UGVs. Attention projections $\mathbf{W}_q^l$, $\mathbf{W}_k^l$, and $\mathbf{W}_v^l$ are set to 128×128 and applied across three layers. The transformer decoder ϕ processes action sequences using two embedding matrices, $\mathbf{E}^a$ and $\mathbf{E}^i$, each mapping inputs to a 128-dimensional space. The decoder contains four layers, where self-attention is parameterized by $\mathbf{W}_q^{l,o}$, $\mathbf{W}_k^{l,o}$, and $\mathbf{W}_v^{l,o}$, and cross-attention is parameterized by $\mathbf{W}_q^{l,c}$, $\mathbf{W}_k^{l,c}$, and $\mathbf{W}_v^{l,c}$, all with dimension 128×128. Finally, a projection matrix $\mathbf{W}^u$ maps the hidden representation $\mathbf{f}_{t-1}^L$ from dimension 128 to the output vocabulary size, which is determined by the specific map configuration.

III. Autonomy Stack Architecture

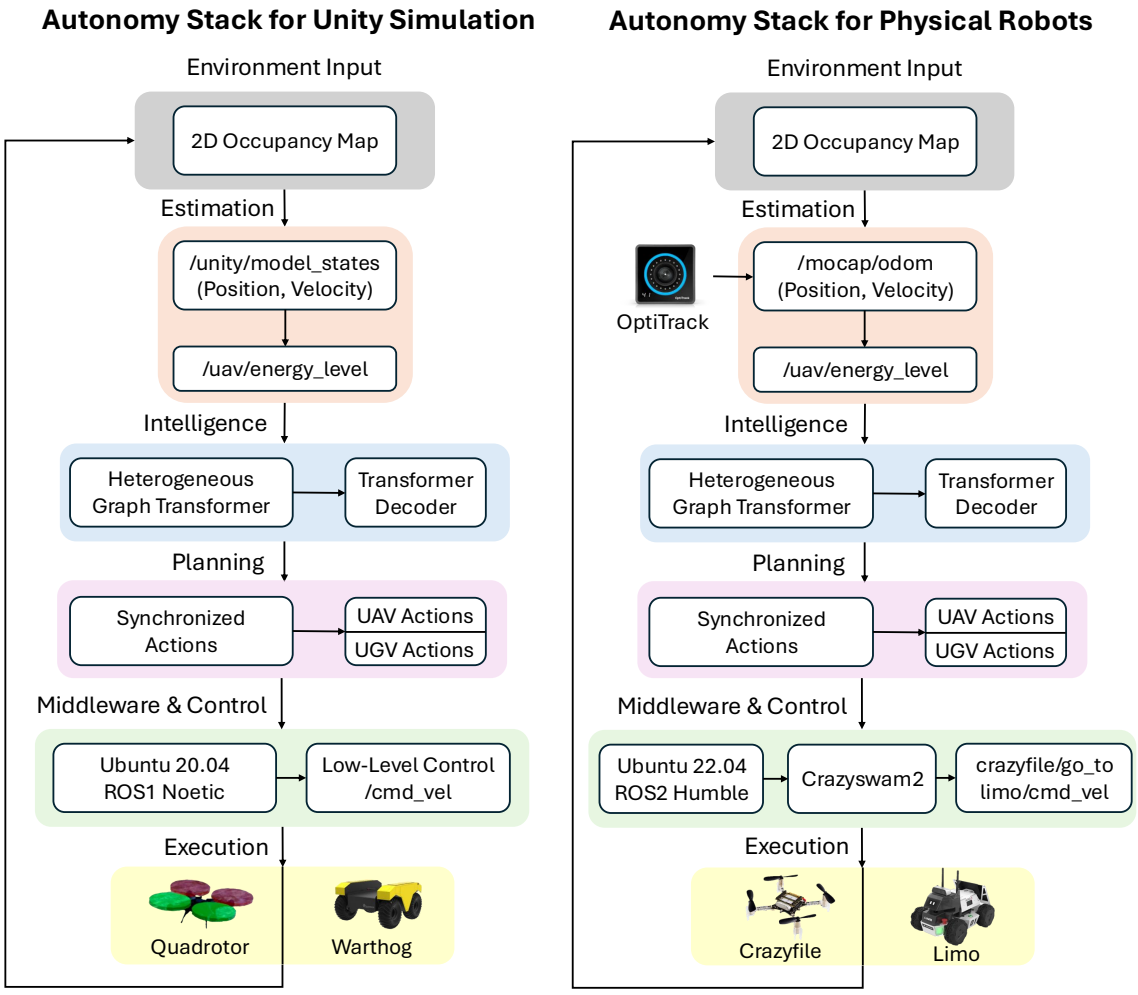

Fig. 1. Autonomy stacks for Unity simulations and physical robots.

We develop two separate autonomy stacks to support Unity-based simulations and physical robot experiments, as illustrated in Figure 1. In both settings, we use namespaces to manage multiple heterogeneous robots. Both stacks are executed on a central workstation equipped with an NVIDIA RTX 4090 GPU and 64 GB RAM.

Unity Simulation Stack. In simulation, we use Ubuntu 20.04 with ROS1 Noetic. The Unity engine provides the 2D occupancy map, ground-truth robot poses, and task and path locations. After CoPCS generates actions for quadrotor UAVs and Warthog UGVs, each robot is controlled through the `/cmd_vel` topic under its own namespace.

Physical Robot Stack. For physical experiments, we use Ubuntu 22.04 with ROS2 Humble. Accurate state estimation is provided by an OptiTrack motion capture system. Each UAV is also assigned an energy-level state to reflect its remaining capacity. Crazyflie UAVs are controlled via the Crazyswarm2 API, where CoPCS outputs are executed through the `go_to` service. Limo UGVs are driven using the `/cmd_vel` topic.