I. INTRODUCTION

In recent years, the unmanned vehicle (UV) has attracted attention due to the advances in communication technology, sensing devices, and computing power. It not only reduces labor costs and brings convenience to life, but more importantly, it can replace some dangerous jobs for humans. Due to these benefits, it has been widely used in many scenarios, such as search and rescue, battlefield, logistics and transportation, and surveillance, etc (cite:UV). Compared with a single UV, multiple UVs can perform more complex tasks and are more robust due to a large number of agents [1]. However, the cost is that the design of such a multi-agent system becomes more intricate as there are more problems to be resolved, such as formation, collision avoidance between agents, task allocation, and cooperation between agents. In addition to the number increasment, a heterogeneous multi-agent system (HMAS) combining various types of UV is also valued (cite:HMAS). Compared with homogeneous multi-agent system, it can adapt to a wider variety of application scenarios. Although there are many types of UVs to make up such a system, Unmanned Aerial Vehicle (UAV) and Unmanned Ground Vehicle (UGV) have been the subject of major recent research because of their availability and applicability (cite:UAV-UGV). Additionally, the complementarity between them also makes such a system more potential. In other words, UAV is widely used in reconnaissance due to the high mobility. However, the carrying capacity of UAV is very low compared to UGV since there is no ground support. In contrast, UGV has higher carrying capacity but is easily restricted by ground obstacles and cannot move at high speed. For these reasons, a hybrid UAVs-UGVs team system will be more appealing.

To construct an unmanned HMAS, the three key capabilities are perception, decision-making and control. Perception is to obtain information through the sensor (e.g., localization or computer vision), decision-making is to make decisions through the sensor information, and control is to execute the decision-making content through the actuator. To limit the scope of this paper, we focus on decision-making and control only. The three main problem of decision-making in an unmanned HMAS are task allocation, path planning and collision avoidance. Task allocation is to optimally assign tasks to each agent, subject to constraints such as agent capabilities, fuel cost, time cost, etc (cite: task allocation). Path planning is to optimally plan paths to each agent, subject to constraints such as agent kinodynamic properties, distance, obstacles collision, etc (cite: path planning). Collision avoidance is to avoid collision with obstacles. Although collision avoidance is often concerned in path planning part, the collision avoidance system is also independently studied because of the requirements for the safety and reliability of the actual system [2].

To discuss an unmanned HMAS more concretely, we consider an UAVs-UGVs team system for search and rescue. For the need for search mobility, we choose quadrotor aircraft as UAV. In order to deal with the complex terrain of the search and rescue environment, biped robot was chosen as the UGV. Although other types of UGVs like wheeled robots and vehicles are easier to handle than biped robot, the high degree of freedom and the compatibility of the human environment still makes it a good candidate of UGV in a search and rescue system.

To the best of the authors' knowledge, most of the literature focus on only one specific problem, and few literatures illustrate the relationship between these problems for an unmanned HMAS. This leads us to propose an system architecture of UAV-robot team system (URTS) for search and rescue usage. The flowchart of decision-making and control process of URTS is given. We divide it into five hierarchical blocks, i.e., tack allocation, path planning, behavior layer, local motion planning and tracking control. First, the UTRS needs to be able to assign different tasks to agents to perform. After a task is assigned, if the task is to reach a target location, a path needs to be planned to reach it. To make agent move on the path, a behavior corresponding to the environment needs to be determined. Then, a local motion corresponding to the behavior of the agent needs to be planned. Finally, a controller must be designed to track the trajectory of the motion. In order to further limit the scope of the study, we will focus on the latter two problems. But to illustrate how the whole system works, the first two problems are also briefly stated.

The local motion planning is the bridge between path planning and tracking control since the path found by path planning algorithm and the path enforced to follow by a controller are not necessarily the same. The reason is that path planning algorithm usually treats the agent as a point, while the actual agent is a mechanical system for the tracking control design. Although there are some literatures (ref:kinodynamic path planning) about path planning with the constraints of kinodynamic of the actual mechanical system, this paper splits the step of path planning into three steps, i.e., (i) path planning, (ii) behavior layer and (iii) local motion planning for clarity. Through this decomposition, we can focus on the local motion design of specific behaviors. The planning of flying motion for UAV and walking motion for robot is studied in this paper, especially the latter. Due to the challenge of the local motion planning of the walking of biped robot, i.e., stable walking parttern generation, it is a popular research topic (cite: walking parttern).

The tracking control is ... The control problem of UAV has been widely studied due to its applicability and low cost (cite: uav control) the control problem of robot FTC disturbance

The contributions of this study are described as follows:

1) contribution 1 ...

The remainder of the paper is organized as follows. In Section II, ...

Notation 1: A^T: transpose of A. (a_n): sequence. (a_{k_n}): subsequence. $|S|$: size of a set S. $\otimes$: Kronecker product. I_n: n-dimension identity matrix. $x(t) \in L_2\left[0, t_f\right]$ if $\int_0^{t_f} x^T(t)x(t)dt < \infty$.

II. PRELIMINARIES OF SEARCH AND RESCUE URTS

The URTS will start with a given search and rescue area, and ends with searching task completed. The URTS is composed of N_j teams and a ground station. Each team contains 1 UAV and $N_i - 1$ robots. Hence, the ith agent in the jth team is denoted as $agent_{i,j}$, where $i = 1, 2, ..., N_i$ and $j = 1, 2, ..., N_j$. The UAVs are chosed as the first agents in each team, i.e., $agent_{1,j}$. Each agent has environmental sensing capability and load capability, while the ground station is responsible for computing and decision-making. Besides, there are communication channels between agents and ground station through wireless network.

To complete search tasks, each team is designed to be responsible for a small area of the overall search area, and each agent will be assigned an appropriate path to cover the area. To complete rescue tasks, whenever a target (e.g., victims or disaster area) is found by the machine vision of nearby agent, the ground station will assign some agents to the location of target. It can be expected that there has multiple search or rescue tasks need to be performed at the same time but there has multiple agents. If each agent can only perform one task at a time, then there exist many feasible way of task allocation. Usually we want this allocation to be optimal, which can be achieved by solving the dynamic task allocation problem.

If a task is to reach a location of certain goal, there must exist multiple feasible paths. Similar to task allocation, we usually want to find an optimal path. Besides, it's necessary to concern about collision between agents or between agents and obstacles in the actual scenario, so each agent will also sense distance-related information about its surrounding and send it back to the ground station. The ground station will combine this information with the goal assigned by task allocation algorithm and make a decision to avoid obstacles and other agents nearby. That is, the path of each agent will be reassigned every moment if needed by solving the dynamic path planning problems.

We need to design some behavior to follow the path found by path planning algorithm especially for agents with complex mechanical systems like robots. Since such a system has a high degree of freedom, there are many ways to follow the same path (e.g., a robot can walk or run to follow the same path). Furthermore, agents in a search and rescue system are not always following a trajectory. Sometimes they need the behavior such as stop to look around, get supplies and put supplies. To meet these needs, a behavioral layer is necessary. The behavior layer gives an appropriate behavior to fulfill an actual requirement. In order to make an behavior, we must design a corresponding motion which is handled by local motion planning. The motion is a desired reference trajectory for an actual mechanical system to follow. Finally, a tracking controller is designed to follow the desired reference trajectory.

To clarify this issue, a more detailed system architecture for each agent is needed. Followed by the concept in [3], a system architecture for an agent used in search and rescue is proposed as shown in Fig. 1. The Simultaneous Localization And Mapping (SLAM) block converts sensor information into the location of agents and an occupy map. The visual object recognition block provides distance information and object information through sensor info. The object information provides agent machine vision that enables it to determine an appropriate behavior (e.g., a robot sees a obstacle and decides to climb through it). The detailed functions of remaining 5 blocks, i.e., Tack Allocation, Path Planning, Behavior Layer, Local Motion Planning and Tracking Control, will be explained in the following sections.

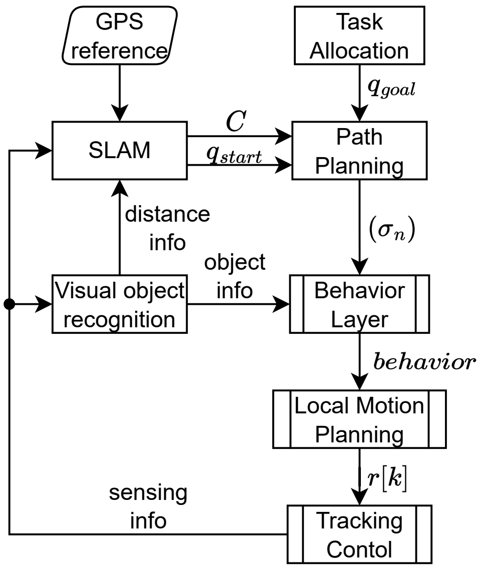

FIGURE 1: The system architecture of agent in URTS. The 2 blocks on left side are used to convert low-level sensor information into high-level information, such as map, start point and object information. The 5 blocks on right side are the flow of an agent performing a search and rescue mission. From top to bottom, it is the decision of the task position, the planning of the path, the decision of the behavior, the planning of the local motion trajectory, and the low-level tracking control

A. TASK ALLOCATION

In the URTS, it can be expected that each agent will be assigned several specific tasks, such as searching a specific area, or delivering supplies to disaster area, etc. However, the number of agent and task is more than one, and each agent has different capabilities (e.g. moving speed or load capacity) and status (e.g. their own position or the amount of supplies carried), and each task has different characteristics (e.g. Urgency, position, amout of supplies needed). Therefore, the results of task allocation can be "good or bad", which makes us want to find the optimal allocation. This problem is refered as Task Allocation Problem or Multi-robot Task Allocation (MRTA) Problem. A problem formulation and problem modeling can be found in [4]. Although there are many different problem definitions and many problem models to solve this issue, the common goal is to find a set of task-agent pairs to achieve a specific cost function. In this paper, we assume that the tasks have been properly assigned and therefore do not further discuss this issue. In other words, every agent knows a destinition it need to go at every moment.

Remark 1: This block is like a commander since it is used to assign task for agents. Thus, if the real search and rescue system has human experts as commanders, he can replace its job or make decisions together with it to maximize the rescue value.

B. PATH PLANNING

After a target point is assigned for each agent, next step is to find a collision-free path from current position to it. There are several path planning algorithm to handle this problem, in this article, a roadmap-based path planning algorithm is used. This method attempts to discretize the search space into interconnected roads and find the path on it. According to the roadmap construction method, it can be divided into deterministic and sample-based. According to the way of pathfinding, it can be divided into multi-query planner and single-query planner [5]. Multi-query planner divides pathfinding into learning phase in which a roadmap is bulit and query phase in which a graph search algorithm is used to find a best path within paths. Some representative planners are Probability Road Map (PRM), Visibility Graph, and Voronoi Diagrams [6]. Relatively speaking, single-query planner completes pathfinding by constructing and querying at the same time, that is to say, the roadmap will be constructed incrementally and toward the goal. Some representative planners such as Rapidly-exploring Random Tree (RRT), Expansice Space Tree (EST), and Ariadne's Clew [5]. However, the environment is dynamic rather static for URTS so some extra structure need to impose on aforementioned planner. Some common dynamic planner also can be found in [5], such like PRM with D* search algorithm, dynamic RRT, and extended RRT.

Furthermore, the constraints imposed by the mechanical structure are needed to consider within pathfinding process mentioned by other literatures but it will be left to Local Motion Planning to deal with. The reason is that URTS is a large-scale system, which makes the environment in which it is located changes more violently and will leads to many corresponding behaviors which path constraints will not necessarily be the same (e.g., curvature constraints of two behavior: running and walking are expected to be different for robot). In this case, Path Planning becomes a higher-level planner which regard agent as a point without kinodynamic constraints while the decision of the behavior and the corresponding motion will be handed over to the next two layers: Behavior Layer and Local Motion Planning.

By treating a roadmap-based path planning algorithm as a black box, the output is a sequence (or sayed waypoints), and the three inputs are current configuration q_{start}, goal configuration q_{goal}, and configuration space $\mathcal{C}$. Current configuration is obtained by GPS, inertial measurement unit, or other locating techniques. Goal configuration is obtained by previous layer: Task Allocation. Configuration space $\mathcal{C}$ is constructed by environment information through sensors of agents online or in advance by human knowledge offline. It's a space contain all possible configuration of agent which composed of free space $\mathcal{C}_{free}$ and obstacle space $\mathcal{C}_{obs}$, where $\mathcal{C} = \mathcal{C}_{free} \cup \mathcal{C}_{obs}$ and $\mathcal{C}_{free} \cap \mathcal{C}_{obs} = \emptyset$. For a simpler explanation of how the URTS works, the following assumption is made.

Assumption 2.1: The locating ability of the URTS is perfect so every agent can know its current configuration q_{start}.

Assumption 2.2: A Task Allocation algorithm is already designed so that every agent can know its goal configuration q_{goal}.

Assumption 2.3: The URTS is supposed to have an perfect real-time mapping ability so a real-time configuration space $\mathcal{C}$ can be obtain.

Assumption 2.4: UAVs do not consider obstacle collisions because there are few obstacles in the air. Hence, the path of UAVs can be directly assigned rather found by planner. Robots do not consider obstacle collisions in the direction perpendicular to the ground.

From above assumption, a path of agent can be expressed as a sequence

$$(\sigma_n), n \in \mathbb{Z} \cap [1, k_f], \sigma_n \in \mathcal{C} \tag{1}$$

where k_f is the time step when reaching goal. Since the path planning is dynamic, (σ_n) is composed of multiple segments actually. Let (σ_{k_n}) be the subsequence of (σ_n), where k_n is the time step when a replanning decision is occured. Then, the segments of path from the result of the replanning in time step k_n can be expressed as sequences $(\sigma_m), m \in \mathbb{Z} \cap [k_n, k_{n+1})$. For agents, the replanning decision can be a goal changing that made by human or Task Allocation. For robot, it can be a collision detecting by a dynamic roadmap-based planner. The results path (σ_n) is passed to next layer, Behavior Layer.

C. BEHAVIOR LAYER

Path Planning tells agents where to go but not how. Take robot as an example, it may walk, run, climb, or jump to follow the path (σ_n) in real scenario. These behaviors with changing position are called "moving" in this paper. Besides, the agents do not always moving. Somtimes they has to suspending to take a action such like getting and putting supplies, rotating in place to collect more environment info, or some unexpected situations occur such like no path found, the robot falls. These behaviors without changing position are called an "action" in this paper. More behaviors can be added so that the agent can have more way to act with environment but there must have a corresponding behavior every moment otherwise the agent will lose control. The sequence of these behaviors be expressed as:

$$(\beta_n), n \in \mathbb{Z} \cap [1, k_f], \beta_n \in \mathcal{B} \tag{2}$$

It is to say that the path (σ_n) is divided into many segments which corresponding to a specific behavior. The set of behavior $\mathcal{B}$ of agent in URTS can be roughly discribed in Fig. 2.

D. LOCAL MOTION PLANNING

After a specific behavior is determined, the next step is to design a motion to achieve that behavior. Local Motion Planning is like Path Planning but its scale is smaller and its resolution and precision must be higher. Collision checking is needed since we consider agent as a point or a cube in Path Planning and it's a real body here. Furthermore, the kinodynamic constraint is handled in this block. Although motion planning and path planning are separated into two blocks, the technologies involved are similar and often the same notion in other literature. Therefore, the output of this block is also a path but with a sampling period which often be referred as a reference trajectory $r(t)$. $r(t)$ describes the position and orientation that needed to be reach by a machine system govern by a dynamic equation. Then, Tracking Control block will track $r(t)$ to really do that behavior. Note that the path and trajectory are distinguished in some literature. Different from path (e.g., (σ_n)), a trajectory (e.g., $r(t)$) has consider the time in physic world. The flowchart of Local Motion Planning block is shown in Fig. 3

In Fig. 3, some basic behaviors of UAV and robot mentioned before are added to illustrate the flow of this block. The terminator, exception, represents an exceptional condition that performs unconsidered behaviors. To limit the scope of this article, we only focus on the motion design of Flying (for UAV) and Walking (for robot) behavior which belong to Moving behavior. The reference path $r[k]$ is a sequence (or say a discrete signal) designed by a specific behavior block. It then convert to reference trajectory $r(t)$, an analog signal, by an A to D convertor.

III. SYSTEM DESCRIPTION OF UAV-ROBOT TEAM SYSTEM

In order to design a reference trajectory $r(t)$ for the motion of UAV and robot, their dynamic models must be given first. After the system description of UAV and robot, the motion design of Flying and Walking as shown in Fig. 4 will be discussed subsequently and separately in the next two subsections.

Before the discussion of the motion design of these two behavior, the following assumption is maded.

Assumption 3.1: The space between obstacles is large enough to eliminate the need for collision checking again, and the average speed of agents is slow enough to ignore dynamic constraints

However, for these two behaviors, there all exists an inevitable kinematic constraint on curvature of local motion. Although an accurate reference trajectory without breaking the curvature constraint can be designed, it is not easy to solve this problem. Additionally, it is not nessasry for these two behaviors in URTS since they are uesd to move from one location to another while the effect of error during moving caused by breaking curvature constraint is relatively insignificant. At the same time, the assumption 3.1 makes sure the error will not causing collision. As an alternative, this problem can be handled by curve fitting which can be regarded as a post-process of the path (σ_n). The post-process will appear in the begining of motion design process of these two behaviors as shown in Fig. 4.

A. MOTION DESIGN OF FLYING OF UAV

A dynamic model about how the UAV moves in the physical world is given first. By Newton-Euler equation, the dynamic model of UAV can be formulated as [7]:

$$\begin{bmatrix} f \\ \tau \end{bmatrix} = \begin{bmatrix} mI & 0 \\ 0 & J \end{bmatrix} \begin{bmatrix} \ddot{X} \\ \ddot{\Theta} \end{bmatrix} + \begin{bmatrix} 0 \\ \dot{\Theta} \times (J\dot{\Theta}) \end{bmatrix} + \begin{bmatrix} f_g \\ 0 \end{bmatrix} + \begin{bmatrix} K_F & 0 \\ 0 & K_\tau \end{bmatrix} \begin{bmatrix} \dot{X} \\ \dot{\Theta} \end{bmatrix} \tag{3}$$

where $J = \begin{bmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{bmatrix}$, $K_F = \begin{bmatrix} K_x & 0 & 0 \\ 0 & K_y & 0 \\ 0 & 0 & K_z \end{bmatrix}$, $K_\tau = \begin{bmatrix} K_{\tau_x} & 0 & 0 \\ 0 & K_{\tau_y} & 0 \\ 0 & 0 & K_{\tau_z} \end{bmatrix}$, $f = \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} = R(\Theta) \begin{bmatrix} 0 \\ 0 \\ F \end{bmatrix}$, $\tau = \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix}$, $X = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$, $\Theta = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}$, $f_g = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix}$, $R(\Theta) = R_z(\psi)R_y(\theta)R_x(\phi)$, $R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$, $R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$, $R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}$. m and J is the mass and inertia matrix of UAV, respectively, τ and f are the total touque and force acting on UAV, respectively, Θ is the Euler angles in body frame, X is the postion of center of mass (CoM) in inertial frame, K_τ and K_F are the aerodynamic damping coefficients, $R(\Theta)$ is the intrinsic rotation matrix from body frame to inertial frame, and F is the total thrust. This model treats the UAV as a mass point and can control the force in the z direction and the touque in the x, y and z direction. Hence, the reference trajectory of UAV $r(t) = [x_r(t), y_r(t), z_r(t), \phi_r(t), \theta_r(t), \psi_r(t)]^T \in \mathbb{R}^6$, where the subscript r denote the reference.

Now, suppose the UAV flying is occured between time step t_1 and t_2, that is, $(\beta_n) = \mathit{flying}, n \in \mathbb{Z} \cap [t_1, t_2]$. The corresponding path $(\sigma_n), n \in \mathbb{Z} \cap [t_1, t_2]$ will be smoothed by linear interpolation and then cubic spline interpolation, which gives the smoothed path $(\sigma'_n), n \in \mathbb{Z} \cap [t_1, t_1 + D(t_2 - t_1)], \sigma'_n \in \mathbb{R}^3$, where $D \in \mathbb{Z}^+$ is the interpolation density. (σ'_n) is the position reference path $\begin{bmatrix} x_r[k] & y_r[k] & z_r[k] \end{bmatrix}^T$. Subsequently, the orientation reference path, row angle $\phi_r[k]$,

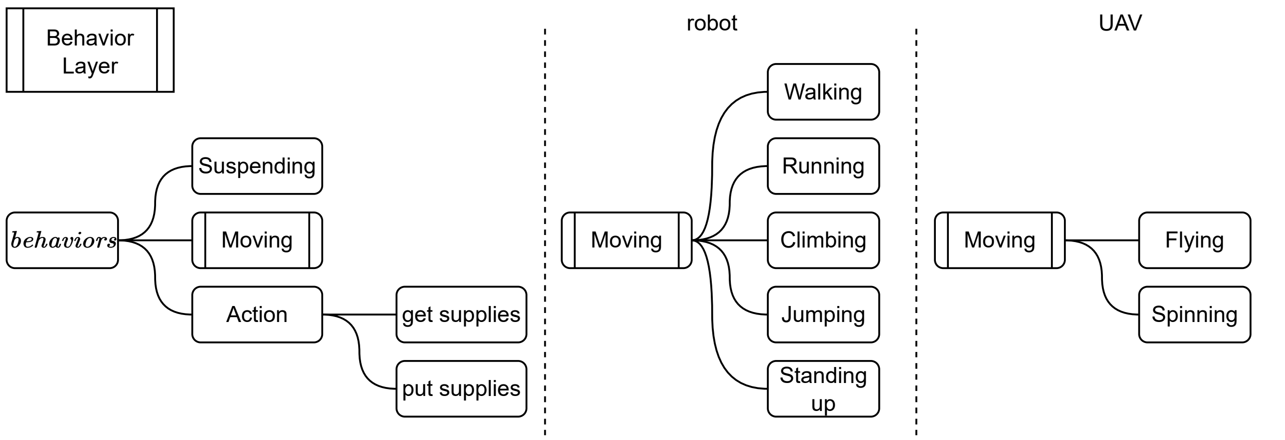

FIGURE 2: The structure of Behavior Layer. The leaf of the tree structure are the possible behaviors an agent can take. The behavior to be take at every moment will be decided in this block.

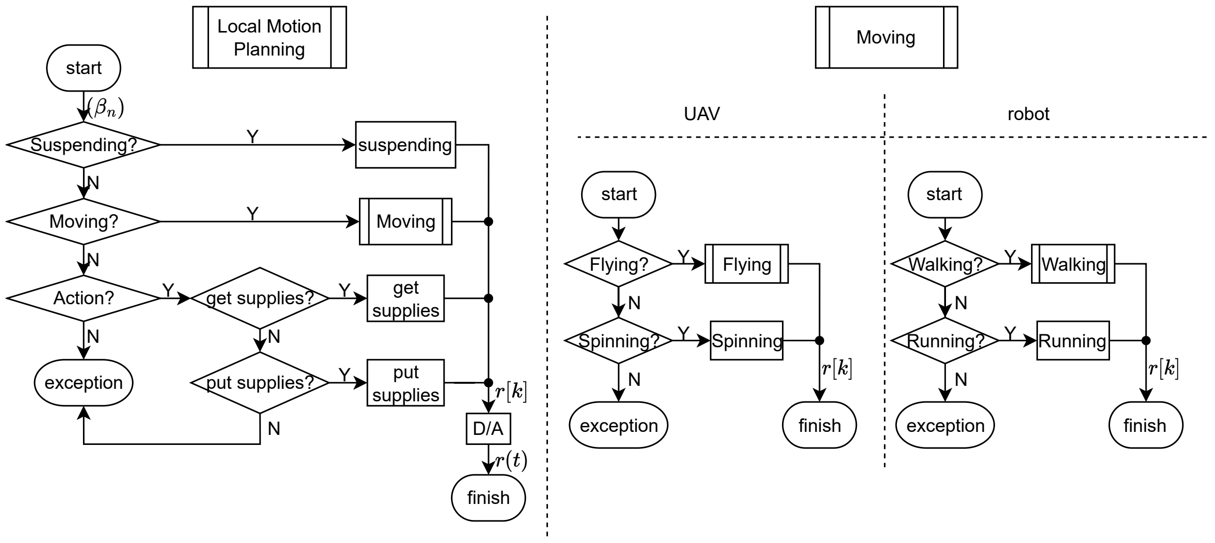

FIGURE 3: The flowchart of local motion planning. The corresponding motion design will be executed according to the behavior determined by the behavior layer.

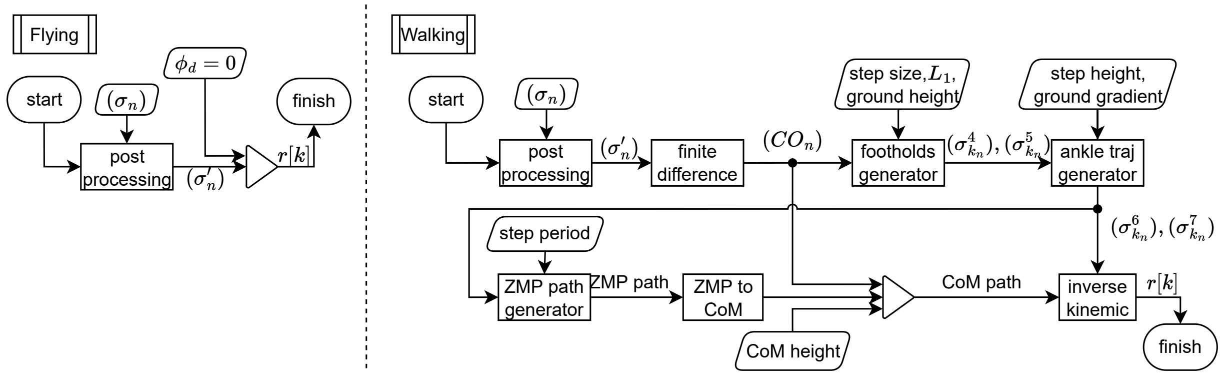

FIGURE 4: The flowchart of Flying and Walking. In both two blocks, a smoothed path is obtained by post-processing first. Then the respective motion of behavior is designed. The final outputs of Flying and Walking are all reference path $r[k]$, but its value and dimension will vary according to the dimension of the system model.

pitch angle $\theta_r[k]$ and yaw angle $\psi_r[k]$ are set. $\phi_r[k]$ is set to zero since no need for spinning when flying. $\theta_r[k]$ and $\psi_r[k]$ cannot be set since UAV is an underactuated system, which will be discussed in the next section. Finally, the reference path $r[k]$ of UAV can be obtained by combining them together, i.e., $r[k] = \begin{bmatrix} x_r[k] & y_r[k] & z_r[k] & \psi_r[k] \end{bmatrix}^T$. The flowchart is shown in Fig. 4.

B. MOTION OF WALKING OF ROBOT

By Lagrange equation, the dynamic model of biped robot can be formulated as:

$$\tau = M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) \tag{4}$$

where $q, \dot{q}, \ddot{q} \in \mathbb{R}^{12}$ are angular position, angular velocity, and angular acceleration vector of revolute joints, $M(q) \in \mathbb{R}^{12\times 12}$ is the inertia matrix, $C(q, \dot{q}) \in \mathbb{R}^{12}$ is the Coriolis and centripetal force vector and $G(q) \in \mathbb{R}^{12}$ is the gravitational force vector. The detailed kinematic and dynamic parameters can be found in the online source [8]. For robot, the reference trajectory $r(t) = q_r(t) \in \mathbb{R}^{12}$ is in the joint space. Furthermore, the walking of biped-robot suffers from the falling problem, i.e, how to find a stable walking partern prevent robot from falling. These make the design of walking motion more difficult. In this paper, a three-dimensional linear inverted pendulum Mode (3D-LIPM) [9] is used to design the walking motion.

Let us define the body frame of robot as $\{\widehat{X_b}, \widehat{Y_b}, \widehat{Z_b}\}$ as in [8]. Taking the forward direction of robot as $\widehat{X_b}$ direction, the left direction as $\widehat{Y_b}$ direction, and the torso direction as $\widehat{Z_b}$ direction in body frame, "Falling" means the moments on the robot in $\widehat{X_b}$ and $\widehat{Y_b}$ direction are not zero. More accuately, the robot will not fall if the zero moment point (ZMP) lies in the support polygon, i.e., the convex hull of face of supported foots. The ZMP in $\widehat{X_b}$ direction can be discribed as [10] ($\widehat{Y_b}$ direction as same form):

$$x_{zmp} = \frac{\sum_{i=1}^{12}(m_i(\ddot{z}_i + g)x_i - m_i\ddot{x}_i z_i - I_{iy}\ddot{\Omega}_{iy})}{\sum_{i=1}^{12} m_i(\ddot{z}_i + g)} \tag{5}$$

It can be seen that ZMP is related to 12 link masses, moment of inertia, positions, accelerations, and angular velocities. This makes the analytical solution of reference $q_r(t)$ impossible. However, an approximate solution can be derived through 3D-LIPM which will find CoM reference first and then obtain $q_r(t)$ by using inverse kinemic (IK) given step size, step height, step period and CoM height. Many researchers have used this method to avoid complex calculations for ZMP of actual robot dynamic model. Although the actual dynamic model is different from 3D-LIPM so there is an error between them, the design process will be more simple. The overall process is shown in Fig. 4.

Following the same step in UAV, the smoothed path (σ'_n) for robot can be obtained at first. For ease of explanation, suppose walking is occured between time step 1 and N, i.e., $n \in \mathbb{Z} \cap [1, N]$. Note that (σ'_n) is not actual CoM reference in robot case since CoM of robot need to "swinging" for balance. Despite that, (σ'_n) tells the robot the position to go so the $\widehat{X_b}$ direction can be obtained by doing finite difference on (σ'_n) due to the expectation that the robot will move forward (rather than sideways or backward). To keep torso upright, the $\widehat{Z_b}$ direction is equal to the z-axis in the inertial frame $\widehat{Z_g}$. Given $\widehat{X_b}$ and $\widehat{Z_b}$, $\widehat{Y_b}$ can be obtained obviously through cross product. The sequence of body frame, i.e., CoM orientation path (CO_n) can be obtained through the above steps.

Remark 2: A frame in $\mathbb{R}^3$ can be determined by given the "position" and "orientation" with respect to a reference frame. That is, given the position and orientation of two joints in a link with known kinematics, the position and orientation of joints between them can be found by IK. Hence, we need to find the position path and orientation path which compose the desired path.

Let us denote the x and y component of σ'_n in (σ'_n) as the sequence $(\sigma^1_n), \sigma^1_n \in \mathbb{R}^2$. The left and right "envelopes", (σ^2_n) and (σ^3_n), of (σ^1_n) with a fixed distance L_1 can be found by (σ^1_n) and (CO_n) through the geometric relation between $(\sigma^i_n), i = 1, 2, 3$, where L_1 is feet width (or shoulder width). Then the x and y component of left and right foothold paths, $(\sigma^2_{k_n})$ and $(\sigma^3_{k_n})$, can be obtained by a given step size, which are the subsequence of (σ^2_n) and (σ^3_n), respectively. Finally, left and right foothold paths $(\sigma^i_{k_n}), \sigma^i_{k_n} \in \mathbb{R}^3, i = 4, 5$ are found by adding z component which is given by ground height.

After foothold paths are obtained, ankle position path can also be obtained by the given step height which is customized by the designer or based on the height of the obstacle to be crossed. Taking the left foothold path as an example, x and y component of the highest position of ankle during stride are set as middle point of two footholds $\sigma^2_{k_m}$ and $\sigma^2_{k_{m+1}}$ where $m \in \mathbb{Z} \cap [1, N-1]$, and the z component are given by step height. By using cubic spline interpolation, we have the left and right ankle position path (σ^6_n) and (σ^7_n). The ankle orientation path is found by gradient of ground. Finally, the ankle path is found by combining the position and orientation path together. So far, the remaining work is to find out the CoM path and then to combine with the ankle path to calculate the joint path through IK.

To obtain CoM postion path (CP_n), ZMP path need to obtain first. ZMP path can be obtained through ankle path since ZMP needs to lie in the support face and ankle path point out when a feet is stand on the ground. Suppose the CoM height z_c of robot is constant when walking, then the robot model can be regarded as an 3D-LIPM:

$$\begin{aligned} \ddot{x} &= \frac{g}{z_c}(x - p_x) \\ \ddot{y} &= \frac{g}{z_c}(y - p_y) \end{aligned} \tag{6}$$

where (x, y) is the position of CoM of the inverted pendulum, g is the gravity acceleration, and (p_x, p_y) is the position of ZMP. Since $z_c, g, (p_x, p_y)$ are given, (x, y) can be solved. To solve the ODE in (6), a method is proposed to convert it to a servo problem [11]:

$$\begin{aligned}\dot{\bar{x}} &= A\bar{x} + Bu \\ p_x &= C\bar{x}\end{aligned} \tag{7}$$

where $\bar{x} = \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix}$, $A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$, $B = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$, and $C = \begin{bmatrix} 1 & 0 & -z_c/g \end{bmatrix}$. Our goal is to find a control input u in order that the output p_x can track a ZMP reference path so that the solution of ODE (6) x, i.e., the CoM position path we want is found. Unlike conventional methods, the problem is solved by optimal control. The system is discretized first and the discrete LQ tracker is employed to achieve the output tracking. The formulation can be found in TABLE 4.4-1 in [12]. The y-direction can be found by same procedure since the dynamic of 3D-LIPM in the x-direction and y-direction is decoupled with same form. The CoM position path (CP_n) then be obtained by combining x, y and z_c.

By combining the CoM orientation (CO_n) and position (CP_n) path, CoM path can be obtained. Finally, the joint path, i.e., reference path $r[k]$ of robot is found by solving IK.

IV. REFERENCE TRACKING CONTROL

After the reference trajectory $r(t)$ is set, the last step is to design a controller for an agent to track it. To analyze the tracking control problem of UAV model in (3) and robot model in (4) together, we represent them by the same form called agent model through some appropriate variable transformations:

$$M(x(t))\ddot{x}(t) + H(x(t), \dot{x}(t)) = u(t) \tag{8}$$

where $u(t) \in \mathbb{R}^n$ is control input vector, $x(t) \in \mathbb{R}^n$ is state vector, $M(\cdot) \in \mathbb{R}^{n\times n}$ is inertia matrix, and $H(\cdot,\cdot) \in \mathbb{R}^n$ is non-inertial force vector. The control law is given as:

$$u(t) = M(r(t))(\ddot{r}(t) + u_{fb}(t)) + H(r(t), \dot{r}(t)) \tag{9}$$

where $M(r(t)), \ddot{r}(t), H(r(t), \dot{r}(t))$ are the feedfoward control terms to canceling system nonlinearity, and $u_{fb}(t)$ is the feedback control law. To make the model more realistic, the following external disturbances encountered in actual scenarios are considered:

1) For agent, there exists coupling effect due to co-channel interference in communication between agents [13].
2) For agent, there exists cyber-attack on communication network between agents and ground station.
3) For agent, there exists sensor noise.
4) For UAV, there exists wind disturbance [14].
5) For robot, there exists ground reaction force [15].

Since the ground station is responsible for the calculation, the calculated control command (9) will be transmitted to the agent through the network channel in URTS. Therefore, the coupling effect due to co-channel interference and the cyber-attack signal will deteriorate the control command. In addition, the wind disturbance and the ground reaction force will apply extra force on an agent (8). Therefore, through appropriate conversion, the above disturbances can be equivalent to an disturbance force $d_1(t)$. The nominal system (8) then be rewrited as the real system:

$$M(x(t))\ddot{x}(t) + H(x(t), \dot{x}(t)) = u(t) + d_1(t) \tag{10}$$

Now, substituting (9) into (10) and subtracting $M(r(t))\ddot{x}(t)$ from the left and right sides, we have:

$$\begin{aligned}&(M(x(t)) - M(r(t)))\ddot{x}(t) + H(x(t), \dot{x}(t)) \\ &= (M(r(t))(\ddot{r}(t) - \ddot{x}(t)) + M(r(t))u_{fb}(t) \\ &+ H(r(t), \dot{r}(t)) + d_1(t)\end{aligned} \tag{11}$$

By Multipling $M(r(t))^{-1}$ from the left and right sides and with some arrangments, we have:

$$\ddot{e}(t) = u_{fb}(t) + f_1(t) \tag{12}$$

where $f_1(t) = M(r(t))^{-1}(-\Delta M - \Delta H + d_1(t))$ is the unknown actuator fault signal. $\Delta M \triangleq M(x(t)) - M(r(t)), \Delta H \triangleq H(x(t), \dot{x}(t)) - H(r(t), \dot{r}(t))$ are the error terms from feedforward compensation, and $e(t) = x(t) - r(t)$ is the tracking error. Let $\boldsymbol{e}(t) = \begin{bmatrix} \int_0^t e(\tau)d\tau & e(t) & \dot{e}(t) \end{bmatrix} \in \mathbb{R}^{3n}$ be the PID tracking error, (12) can be rewrited as:

$$\dot{\boldsymbol{e}}(t) = A\boldsymbol{e}(t) + B(u_{fb}(t) + f_1(t)) \tag{13}$$

where $A = A_0 \otimes I_n, B = B_0 \otimes I_n$ with $A_0 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, B_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$.

Through above analysis, the tracking control problem of the nonlinear system with external disturbance (10) is transformed into the fault tolerance stable control problem of linear system (13). The remaining step is to design an appropriate feedback control law $u_{fb}(t)$ to make the linear system stable. In a real system, the feedback information is measured by sensor, i.e, the state $\boldsymbol{e}(t)$ in (13) is unavailable. At the same time, the external disturbances on sensor also need to be considered as mentioned before. Since the sensor information (14) will transmitted back to the ground station for calculating control command through the network channel in URTS, not only the sensor noise but also the coupling effect due to co-channel interference and the cyber-attack signal are concerned. Suppose the total effect can be equivalent to an unknown sensor fault signal $f_2(t) \in \mathbb{R}^n$ with same size as $f_1(t)$, the measurement output equation can be described as:

$$y(t) = C\boldsymbol{e}(t) + B_2 f_2(t) \tag{14}$$

where $y(t) \in \mathbb{R}^l$ is the output vector, $C \in \mathbb{R}^{l\times 3n}$ is the output matrix, and $B_2 \in \mathbb{R}^{l\times n}$ is the input matrix of $f_2(t)$. By putting (13) and (14) together, we have the following system:

$$\begin{aligned}\dot{\boldsymbol{e}}(t) &= A\boldsymbol{e}(t) + B(u_{fb}(t) + f_1(t)) \\ y(t) &= C\boldsymbol{e}(t) + B_2 f_2(t)\end{aligned} \tag{15}$$

To deal with the fault signals, an active fault tolerance control (FTC) is introduced [16]. First, we need a model of the fault signal. In general, constructing a model for an unknown signal is impossible since there is no information about it. Hence, the following assumption is maded:

Assumption 4.1: The first $p-1$ derivatives of the fault signals $f_i(t), i=1,2$ all exist and are continuous, i.e., $f_i(t) \in C^{p-1}$. Under the assumption, now we can construct a "smooth" model by finite difference method. To obtain a state-space representation, the derivative of the fault signals $\dot{f}_i(t)$ with uniform grid h are needed, which can be expressed as:

$$\dot{f}_i(t) = \sum_{k\in Z_0} \frac{a_k f_i(t-kh)}{h} + R_{p-1}(t), Z_0 \subset \mathbb{Z} \tag{16}$$

where $R_{p-1}(t) \in O(h^{p-1})$ denotes the remainder term, and $p = |Z_0| > 1$ is the accuracy of difference. The derivative of $f_i(t-kh), k \in Z_0$ in (16) is also needed. For the convenience of analysis, let us choose $Z_0 = \{0,1,\ldots,p-1\}$. By changing index k to j, the derivative of $f_i(t-jh), k \in Z_0$ can be obtained by (16):

$$\dot{f}_i(t-jh) = \sum_{k=0}^{p-1} \frac{a_{j,k} f_i(t-kh)}{h} + \epsilon_j(t), j=0,1,\ldots,p-1 \tag{17}$$

where $\epsilon_j(t) \in O(h^{p-1})$. By arranging $\dot{f}_i(t-jh)$ into a state-space representation, we have:

$$\dot{F}_i(t) = A_i F_i(t) + v_i(t) \tag{18}$$

where $v_i(t) = \begin{bmatrix} \epsilon_0^T(t) & \epsilon_1^T(t) & \ldots & \epsilon_{p-1}^T(t) \end{bmatrix}^T$, $F_i(t) = \begin{bmatrix} f_i^T(t) & f_i^T(t-h) & \ldots & f_i^T(t-(p-1)h) \end{bmatrix}^T$, and $A_i = (a_{j,k}) \otimes I_n$. Then, the fault signals $f_i(t), i=1,2$ are the output of the system in (18)

$$f_i(t) = C_i F_i(t) \tag{19}$$

where $C_i = \begin{bmatrix} 1 & 0 & \ldots & 0 \end{bmatrix} \otimes I_n$. Sub. (18) and (19) into (15), we get:

$$\begin{aligned} \dot{\bar{e}}(t) &= \bar{A}\bar{e}(t) + \bar{B}u_{fb}(t) + \bar{v}(t) \\ y(t) &= \bar{C}\bar{e}(t) \end{aligned} \tag{20}$$

where $\bar{e}(t) = \begin{bmatrix} \boldsymbol{e}(t) \\ F_1 \\ F_2 \end{bmatrix}$, $\bar{A} = \begin{bmatrix} A & BC_1 & 0 \\ 0 & A_1 & 0 \\ 0 & 0 & A_2 \end{bmatrix}$, $\bar{B} = \begin{bmatrix} B \\ 0 \\ 0 \end{bmatrix}$, $\bar{C} = \begin{bmatrix} C & 0 & C_2 \end{bmatrix}$, and $\bar{v}(t) = \begin{bmatrix} 0 \\ v_1(t) \\ v_2(t) \end{bmatrix}$.

Remark 3: The equation (16) is derived by Taylor expansion, so assumption 4.1 is nessasry. Unlike previous work, a finite-difference-method-based modeling method is proposed, which reduce the approximation error between origin fault signal and its smooth model. Besides, the fault signals $f_i(t)$ are assumed to be of differentiability class C^{p-1} rather than bounded, which will make the proposed method more general for practical scenario.

Since the fault signals become a state variable of the augment system in (20), a Luenberger observer is proposed to estimate them and origin state simutaniously to achieve active FTC:

$$\begin{aligned} \dot{\hat{\bar{e}}}(t) &= \bar{A}\hat{\bar{e}}(t) + \bar{B}u_{fb}(t) + L(y(t) - \hat{y}(t)) \\ \hat{y}(t) &= \bar{C}\hat{\bar{e}}(t) \end{aligned} \tag{21}$$

Assumption 4.2: The augmented system (20) is observable. The PID FTC law is given as:

$$u_{fd}(t) = K\hat{\bar{e}}(t) \tag{22}$$

where $K = \begin{bmatrix} K_I & K_P & K_D & K_{F_1} & K_{F_2} \end{bmatrix}$ is the total control gain, K_P, K_I, K_D are the PID control gain for position tracking error $e(t)$, and $K_{F_i}, i=1,2$ are the fault control gain. Let us define the estimation error $\tilde{e} = \bar{e}(t) - \hat{\bar{e}}(t)$, then an new augmented system can be obtained from (20), (21) and (22):

$$\dot{\tilde{x}}(t) = \tilde{A}\tilde{x}(t) + \tilde{v}(t) \tag{23}$$

where $\tilde{x}(t) = \begin{bmatrix} \bar{e}(t) \\ \tilde{e}(t) \end{bmatrix}$, $\tilde{A} = \begin{bmatrix} \bar{A}+\bar{B}K & -\bar{B}K \\ 0 & \bar{A}+L\bar{C} \end{bmatrix}$, and $\tilde{v}(t) = \begin{bmatrix} \bar{v}(t) \\ 0 \end{bmatrix}$

In order to enable the designed control gain K and observer gain L to achieve a specific stablized performance for the system (23) under the disturbance $\bar{v}(t)$, the H_∞ observer-based stabilized control performance below a prescribed disturbance attenuation level ρ^2 is given as:

$$\begin{aligned} &H_\infty(K,L) \\ &= \frac{\int_0^{t_f} (\tilde{x}^T(t)Q\tilde{x}(t) + u^T(t)Ru(t))dt - V(\tilde{x}_0)}{\int_0^{t_f} \tilde{v}^T(t)\tilde{v}(t)dt} \leq \rho^2 \end{aligned} \tag{24}$$

where t_f is the final time, $Q > 0$ is the weighting matrix of state and estimation error, $R > 0$ is the weighting matrix of control effort, $V(\tilde{x}_0)$ is the initial condition effect on augmented system(23), and $\tilde{v}(t)$ is the total disturbance needs to be attenuated. If we can find the control gain K and observer gain L such that (24) holds, then the effect of total disturbance $\tilde{v}(t)$ on tracking and estimation error $\tilde{x}(t)$ can be attenuated to a prescribed level ρ^2 from the viewpoint of energy. To find these gains, the following theroem is given.

Theorem 1: If

$$\begin{bmatrix} M_{11} & -\bar{B}Y1 & W_1\sqrt{Q_1} & 0 \\ * & M_{22} & 0 & P_2 \\ * & * & -I & 0 \\ * & * & * & -\rho^2 \end{bmatrix} \leq 0 \tag{25}$$

where $M_{11} = \bar{A}W_1 + \bar{B}Y_1 + (\bar{A}W_1 + \bar{B}Y_1)^T + \frac{1}{\rho^1}I$, $M_{22} = P_2\bar{A} + Y_2\bar{C} + (P_2\bar{A} + Y_2\bar{C})^T + Q_2$. Then system (23) achieve the H_∞ observer-based stabilized control performance (24).

Proof. proof... ■

Although the control gain and observer gain that satisfy with the specification can already be found by Theorem 1, the calculation speed can be further improved by reducing the dimensionality of the LMI (25). Observing the matrices A, B, C, B_2 in linear system (15), it can be further split into n subsystems if the matrices C, B_2 have the same form to A, B, i.e., $C = C_0 \otimes I_n, B_2 = B_{2,0} \otimes I_n$ where $C_0 \in \mathbb{R}^{l\times 3}, B_{2,0} \in \mathbb{R}^{l\times 1}$. Due to the fact that $f_1(t) \in \mathbb{R}^n, f_2(t) \in \mathbb{R}^n$, the n subsystems are obtained from (15):

$$\begin{aligned} \dot{\boldsymbol{e}}_i(t) &= A_0\boldsymbol{e}_i(t) + B_0(u_{fd,i}(t) + f_{1,i}(t)) \\ \boldsymbol{y}_i(t) &= C_0\boldsymbol{e}_i(t) + B_{2,0}f_{2,i}(t) \end{aligned} \tag{26}$$

where subscript $i = 1, 2, ..., n$ denote the element in the corresponding vector. By Theorem 1 again, the form shows that the subsystems (26) can achieve the H_∞ observer-based stabilized control performance. Let K_i be the control gain of the ith subsystems, the origin control gain K of the origin agent system can be obtained by $K = \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix}^T, v_i = K_i^T \otimes \mathbf{e}_i$ where $\mathbf{e}_i$ is standard unit column vectors in $\mathbb{R}^n$. The origin observer gain L can be obtained in the same way. In this case, the calculate speed can be improved since the dimensionality is decrease.

Remark 4: This method can regard as designing the control and observer gain to each state variable, which is a common control method in practice. However, the gains are not directly adjusted but indirectly designed through prescribed specifications (24). Besides, $K_i, i = 1, 2, \dots, n$ do not have to be the same value (so do $L_i, i = 1, 2, \dots, n$). Dependent on the actual system situation, the appropriate L_i and K_i can be designed for each state variable by adjusting the weighting matrix Q_i, R_i and attenuation levels ρ_i for each subsystem.

Although the design of the agent controller and observer has been completed, the trajectory design of UAV is not done yet. Since the UAV is an underactuated system, the "virtual" control input for UAV $u(t) = \begin{bmatrix} f_x & f_y & f_z & \tau_x & \tau_y & \tau_z \end{bmatrix}^T \in \mathbb{R}^6$ obtained from Theorem 1 needs to convert to actual control input $u'(t) = \begin{bmatrix} F & \tau_x & \tau_y & \tau_z \end{bmatrix}^T \in \mathbb{R}^4$. At the same time, the extra two degrees of freedom are used to find the remaining two reference trajectories ϕ_r and θ_r through inverse dynamics. That is to say, we want to find a total force F and its angles ϕ_r, θ_r and ψ_r from the three component forces f_x, f_y and f_z. By substituting $\Theta = \begin{bmatrix} \phi_r \\ \theta_r \\ \psi_r \end{bmatrix}$ into $\begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} = R(\Theta) \begin{bmatrix} 0 \\ 0 \\ F \end{bmatrix}$ from UAV dynacmis in (3), the 3 unknown variables F, ϕ_r and θ_r can be found from these 3 equations because f_x, f_y, f_z and ψ_r are given.

The overall flowchart of Tracking Control is shown in Fig. 5. The reference generator is used to compute the actual reference trajectory $r(t)$ and control input $u(t)$ according to the result of Local Motion Planning $r'(t)$ and Theorem 1 $u'(t)$. The reference generator for robot is an identity function since it is a fully actuated system. Passing $r(t)$ through the integrator and differentiator, we get $\boldsymbol{r}(t) = \begin{bmatrix} \int_0^t r(\tau)d\tau & r(t) & \dot{r}(t) \end{bmatrix}$. $\boldsymbol{r}(t)$ then pass to sensor to calculate the PID tracking error $\boldsymbol{e}(t)$. The sensor measure not only the agent's own information (e.g., position, velocity or $\boldsymbol{e}(t)$) but environmental information. The environmental information is passed back to the high-level block for positioning, mapping and object recognition. The measurement output $y(t)$ is input to observer to get the estimation $\hat{e}(t)$ for feedback control. The differentiation of $\boldsymbol{r}(t)$, $\dot{\boldsymbol{r}}(t)$, is input to controller for feedforward control.

V. SIMULATION RESULTS

VI. CONCLUSION

REFERENCES

[1] Ali Dorri, Salil S. Kanhere, and Raja Jurdak. Multi-agent systems: A survey. IEEE Access, 6:28573–28593, 2018.

[2] Jawad N. Yasin, Sherif A. S. Mohamed, Mohammad-Hashem Haghbayan, Jukka Heikkonen, Hannu Tenhunen, and Juha Plosila. Unmanned aerial vehicles (uavs): Collision avoidance systems and approaches. IEEE Access, 8:105139–105155, 2020.

[3] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. IEEE Transactions on intelligent vehicles, 1(1):33–55, 2016.

[4] Alaa Khamis, Ahmed Hussein, and Ahmed Elmogy. Multi-robot Task Allocation: A Review of the State-of-the-Art, pages 31–51. Springer International Publishing, Cham, 2015.

[5] Mohamed Elbanhawi and Milan Simic. Sampling-based robot motion planning: A review. Ieee access, 2:56–77, 2014.

[6] Yuanchang Liu and Richard Bucknall. A survey of formation control and motion planning of multiple unmanned vehicles. Robotica, 36(7):1019–1047, 2018.

[7] Francesco Sabatino. Quadrotor control: modeling, nonlinearcontrol design, and simulation, 2015.

[8] M.Y.Lee B.S.Chen, Y.Y.Tsai. online supplementary file. https://www.dropbox.com/s/qiv63y5hw7vrubp/SupplementFileRobot.pdf.

[9] Shuuji Kajita, Osamu Matsumoto, and Muneharu Saigo. Real-time 3d walking pattern generation for a biped robot with telescopic legs. In Proceedings 2001 ICRA. IEEE international conference on robotics and automation (Cat. no. 01ch37164), volume 3, pages 2299–2306. IEEE, 2001.

[10] Qiang Huang, Kazuhito Yokoi, Shuuji Kajita, Kenji Kaneko, Hirohiko Arai, Noriho Koyachi, and Kazuo Tanie. Planning walking patterns for a biped robot. IEEE Transactions on robotics and automation, 17(3):280–289, 2001.

[11] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In 2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422), volume 2, pages 1620–1626 vol.2, 2003.

[12] Frank L Lewis, Draguna Vrabie, and Vassilis L Syrmos. Optimal control. John Wiley & Sons, 2012.

[13] Bor-Sen Chen, Yueh-Yu Tsai, and Min-Yen Lee. Robust decentralized formation tracking control for stochastic large-scale biped robot team system under external disturbance and communication requirements. IEEE Transactions on Control of Network Systems, 8(2):654–666, 2021.

[14] Jun Yang, Cunjia Liu, Matthew Coombes, Yunda Yan, and Wen-Hua Chen. Optimal path following for small fixed-wing uavs under wind disturbances. IEEE Transactions on Control Systems Technology, 29(3):996–1008, 2021.

[15] Bor-Sen Chen, Chia-Chou Wu, and Yen-Wen Chen. Human walking gait with 11-dof humanoid robot through robust neural fuzzy networks tracking control. International Journal of Fuzzy Systems, 15(1), 2013.

[16] Bor-Sen Chen, Min-Yen Lee, Wei-Yu Chen, and Weihai Zhang. Reverse-order multi-objective evolution algorithm for multi-objective observer-based fault-tolerant control of t-s fuzzy systems. IEEE Access, 9:1556–1574, 2021.

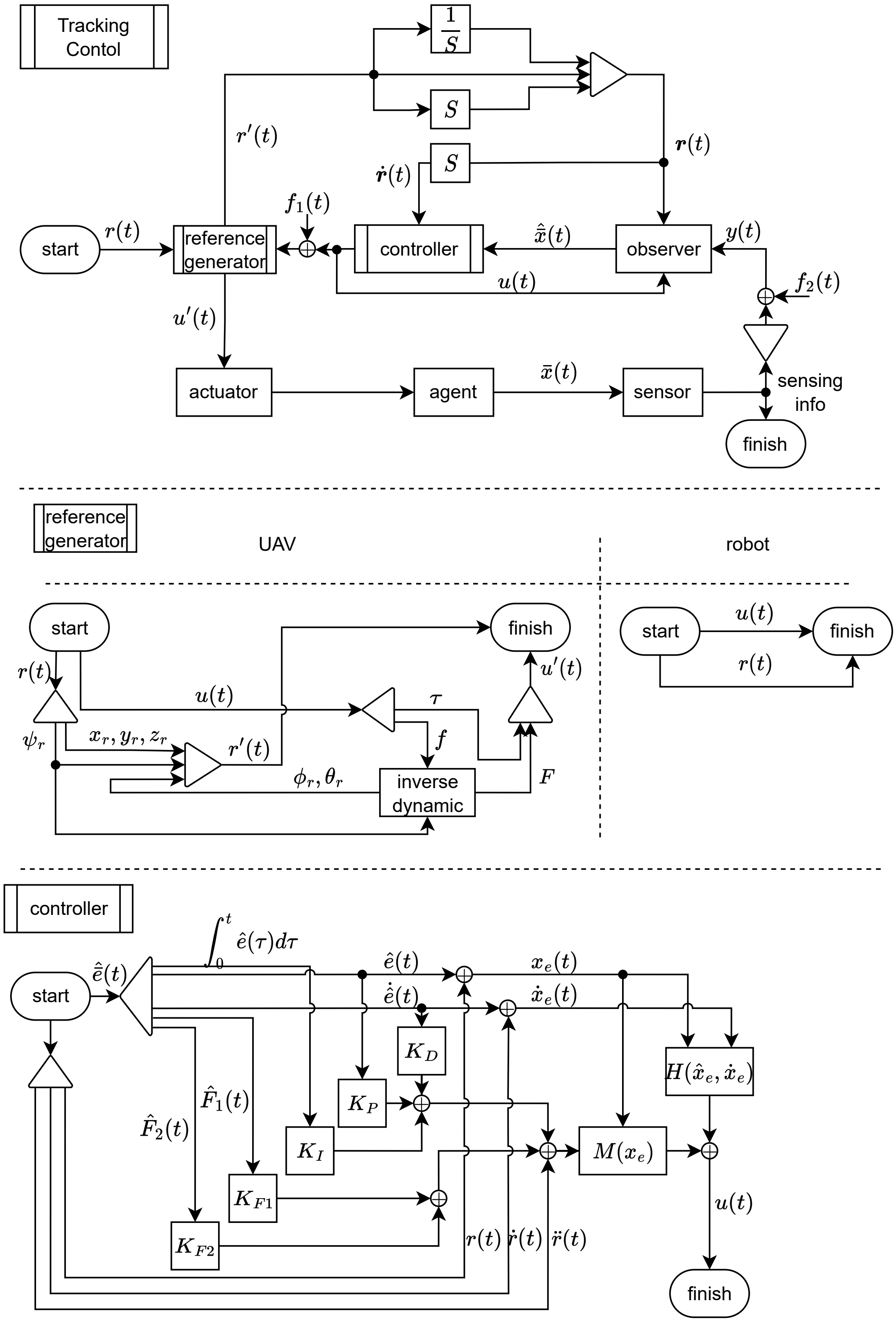

FIGURE 5: The flowchart of tracking control. The control scheme of UAV and robot is only different from the block, reference generator. The reason is that the underactuated nature of the UAV imposes the limitation on the control input and reference trajectory. By introducing this block, a general feedforward-linearized PID FTC for an agent in URTS is proposed.