Mission Planning and Execution in Heterogeneous Teams of Aerial Robots supporting Power Line Inspection Operations

Álvaro Calvo[1], Giuseppe Silano[2], and Jesús Capitán[1]

***Abstract*— A software architecture aimed at coordinating a team of heterogeneous aerial vehicles for inspection and maintenance operations in high-voltage power line scenarios is presented in this paper. A hierarchical approach deals with high-level tasks by planning and executing complex missions requiring vehicles to support human operators. A resource-constrained problem allows distributing tasks among the team taking into account vehicles' capabilities and battery constraints. Besides, Behavior Trees (BTs) are in charge of mission execution, triggering replanning operations in case of unforeseen events, such as vehicle faults or communication drop-outs. The feasibility and validity of the approach are showcased through realistic simulations achieved in Gazebo.**

***Index Terms*— Task planning and execution; Multi-UAV systems; Behavior Trees; Power line inspection and maintenance.**

I. Introduction

Energy demand has increased significantly over the last decades. In order to keep up with this pace, inspection and maintenance operations in electric power lines and related infrastructures are becoming of uppermost importance for supply companies, as a way to prevent power outages and mitigate economic losses. Nowadays, these operations are usually scheduled periodically and carried out by experienced working crews, who gather inspection data and repair/replace the damaged parts on active lines using manned helicopters. However, this procedure is highly risky, as humans need to operate at height, under windy conditions, and in hazardous environments.

Therefore, there is a strong interest of power suppliers in finding appropriate technologies [1] to increase safety and efficiency in these maintenance activities. Unmanned Aerial Vehicles (UAVs) constitute a promising solution to automate inspection operations [2], as they can work in hazardous places, inspect remote locations of difficult access, and monitor human operation for safety purposes. The idea of using a team of heterogeneous UAVs cooperating and performing various tasks to support a human crew on site is even more appealing (see Fig. 1). However, this problem is challenging for several reasons: (i) UAVs have limited time of flight and payload, which forces them to schedule recharging operations for a longer endurance; (ii) heterogeneity prohibits the arbitrary exchange of one UAV for another, which makes coordination for proper mission accomplishment more complex; and (iii) the inspection needs to be performed in dynamic settings, where UAVs or communication could fail. Therefore, there is a need for cooperative multi-UAV architectures where heterogeneous vehicles can efficiently compute and execute plans for long inspection missions, extending their flight autonomy and reacting online to unforeseen events, such as communication drop-outs or vehicle's breakdowns or faults.

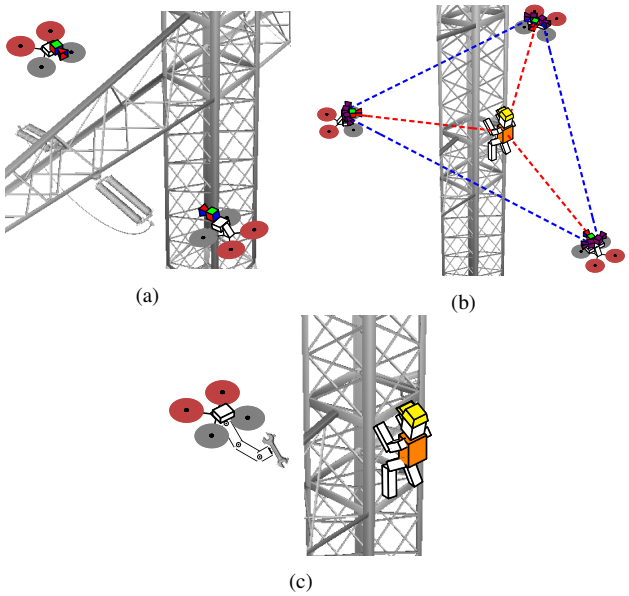


Fig. 1: Heterogeneous UAVs supporting inspection and maintenance operations. From left to right: *inspection*, *monitoring*, and *delivery* scenarios.

In this paper, we propose a software architecture for planning and execution of inspection and maintenance missions with a heterogeneous fleet of multi-rotor UAVs to support human crews in power lines assessment operations. In terms of planning, the required tasks are allocated to different UAVs depending on their capabilities, and recharging tasks are scheduled for longer operation. Given the dynamic and uncertain conditions of the environment where the mission takes place (e.g, the number of available UAVs, the pending tasks, actual battery consumption, tasks' duration, etc.), the execution of the mission is steadily monitored in order to react to unplanned events by recomputing the task assignment online. The software architecture extends and complements our prior work [3], where we integrated the UAV's low-level capabilities taking the task planning part for granted.

This work is funded by the the European Union's Horizon 2020 research and innovation programme AERIAL-CORE under grant agreement no. 871479.

[1]Álvaro Calvo and Jesús Capitán are with the GRVC Robotics Laboratory, University of Seville, 41092 Seville, Spain, (email: {acalvo1, jcapitan}@us.es).

[2]Giuseppe Silano is with the Faculty of Electrical Engineering, Department of Cybernetics, Czech Technical University in Prague, 12135 Prague, Czech Republic, (email: giuseppe.silano@fel.cvut.cz).

A. Related work

In a multi-robot context, *mission planning* consists of deciding which tasks to allocate to each robot and then building plans with the assignment. *Mission execution* carries out those plans and monitors them for mission accomplishment. Both problems have been widely studied in the literature over years [4], [5] and they can be mainly grouped into two classes, depending on whether the task allocation problem is addressed in a centralized or decentralized fashion. Centralized solutions, in the form of constrained optimization problems, allow to retrieve the best schedule for each robot [6], [7]. However, these solutions suffer from the high computational burden required to solve the optimization problem. On the other hand, decentralized solutions [8], [9] can address the computational load by sharing the problem complexity over the robots. However, such an approach is formulated at the expense of an increase in the inter-robot communication or state estimation, which may be unfeasible in some applications. Therefore, centralized approaches can be more suitable for scenarios with a bounded number of robots, mainly if communication network reliability and compliance with safety requirements are among the mission objectives.

Centralized auction algorithms [10] and consensus-based methods [11] are commonplace. Some of these works present solutions addressing heterogeneity in the robots' capacities [6], [11]. Whereas, in missions where the tasks are placed at different locations to be visited, routing problem formulations are preferred. However, the latter easily ends up being combinatorial NP-hard problems, so heuristic techniques are usually applied to retrieve the solution within a reasonable time [12], [13]. Alternatives consider the inclusion of temporal constraints and uncertainties in the tasks' descriptions [4]. In this regard, Temporal Logic [14] can deal with the tasks allocation when complex mission requirements need to be met. However, these problems easily become very complex, being non-convex *min-max* optimization problems [15]. On the other hand, some works propose multi-UAV task allocation frameworks integrating mission planning and execution [16]–[18]. In [16], heterogeneous teams with ground and aerial vehicles are combined so that UAVs can land by carrying out recharging operations and thus extending their autonomy. In [17], event-triggered replanning is considered during the mission execution. ROSPlan [18] focuses on deterministic complex planning problems.

Finally, regarding the encoding of the UAV behavior during mission execution, Finite State Machines (FSMs) are still the most widespread option [19], although recently the use of Colored Petri nets [20] have been also proposed. Among these, Behavior Trees (BTs) are gaining momentum [21], thanks to the advantages in terms of behavior composability and reusability, fault tolerance, and parallel task execution.

B. Contributions

In this paper, we propose a software architecture for mission planning and execution of inspection and maintenance tasks with a heterogeneous team of UAVs for power lines assessment missions. Our approach is based on a centralized *High-Level Planner*, that assigns tasks to the UAVs and schedules recharging operations in between based on operator's requests, along with a distributed *Agent Behavior Manager*, in charge of mission execution on board each vehicle. Specifically, in Section II we define an application-driven problem for supporting human crews during inspection missions. While, in Section III we propose a software architecture for multi-UAV mission planning and execution. The software stack is released as open-source[1] making it possible to go though any part of the framework and replicate the obtained results. Our main contributions are as follows:

- We propose a heuristic planner (Section III-A) that can cope with UAVs heterogeneous capabilities and battery constraints, allocating tasks and scheduling recharging operations in between.
- A distributed management component based on BTs (Section III-B) is in charge of monitoring the tasks execution and requesting an online replanning in case of system's breakdowns or failures (e.g., a UAV runs out of battery).
- Software-In-The-Loop (SITL) simulations have been carried out in realistic simulation scenarios showcasing the performance and feasibility of our methods (Section IV) and providing insight into future directions (Section V). Benefits in terms of computation time for planing missions are also provided.

II. Problem Description

Three tasks of interest are considered: (i) *inspection*, where a fleet of UAVs carries out a detailed investigation of power equipment on its own, assisting human operators in acquiring views of the power tower that are not easily accessible, as depicted in Fig. 1a; (ii) *monitoring*, where a formation of UAVs provides to the supervising team a view of the humans working on the power tower to monitor their status and ensure their safety, as shown in Fig. 1b; and (iii) *delivery*, where a UAV equipped with loading capabilities interacts with a human worker to deliver a tool, as reported in Fig. 1c. Some tasks only require a single UAV, while others involve several UAVs who must work together to achieve a common objective. Moreover, depending on their capabilities, some UAVs can perform different types of tasks, by serving as both an inspection and monitoring UAV, whereas those with capabilities for physical interaction are the only ones that can perform delivery tasks.

We assume that the UAVs operate in a known environment represented by a previously acquired map, including the position of the power towers and lines. Besides, precise algorithms for UAV localization and navigation are taken for granted. Specifically, the UAVs in the team are endowed with a proper set of hardware equipment and *low-level controllers* for the execution of the defined tasks. Moreover, the UAVs can detect human gestures [22] that are used to provide

[1] https://github.com/grvcTeam/aerialcore_planning

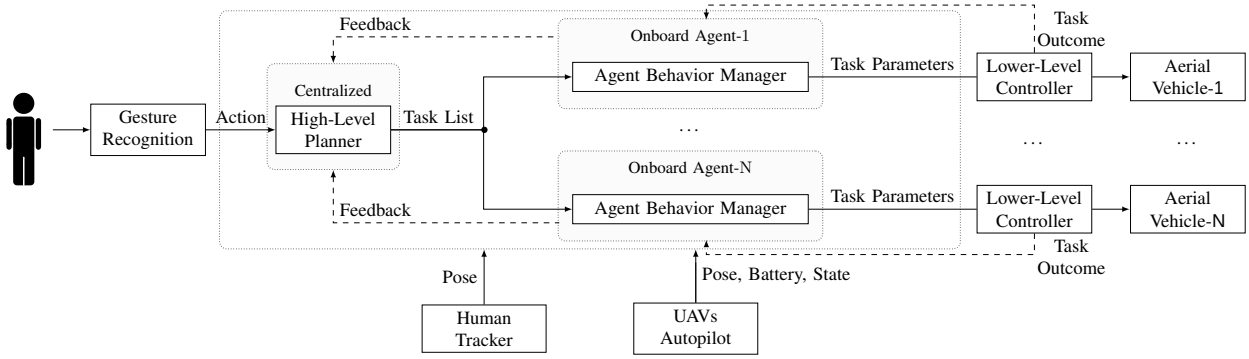


Fig. 2: The software architecture. Arrows represent the data exchange among blocks, while dotted boxes are used to identify the two layers composing the architecture.

high-level actions, such as requests for new tasks or new parameters for a previously requested task.

In the *inspection* operations, the low-level controller encodes the mission execution as a trajectory planning problem where the vehicles need to move from an initial position and through a sequence of target points, while avoiding obstacles and maintaining a safety distance between them [14]. In the *monitoring* operations,the corresponding low-level controller solves a formation control problem where the vehicles need to keep the human worker within the camera frame during the entire operation, providing complementary views from multiple angles [23], [24]. In both tasks, quadrotor UAVs support maintenance operations. Last, in *delivery* missions the UAV is equipped with hardware to transport and deliver tools to a human worker. The corresponding low-level controller implements human-aware motion planning and control algorithms [25] for a safe interaction with the worker.

Given these settings, the objective is to coordinate the heterogeneous fleet of UAVs to perform all the operator's actions while complying with drones' battery constraints and unforeseen events that may occur during the mission execution, such as the arrival or a failure of an action and a UAV running out of battery or communication drop-outs.

III. Software Architecture

The software architecture is organized into two layers: the *High-Level Planner* (Section III-A) and the *Agent Behavior Manager* (Section III-B). The former is placed at a ground station, while the latter run onboard the UAVs. Figure 2 describes the overall software architecture. The *High-Level Planner* interacts with the human crew through a *Gesture Recognition* block [22]. Such a block encodes the human gestures into actions for the fleet of aerial vehicles. Hence, the *High-Level Planner* outputs a set of tasks each vehicle has to execute to fulfill the mission objectives. Then, N-instances of the *Agent Behavior Manager* take care of extracting the necessary parameters from the assigned tasks (e.g., the waypoints to inspect, the human worker to monitor or to whom deliver a tool, etc.) and coordinate the mission execution based on the inputs provided by the *Human Tracker* [22] (which provides the worker position) and *UAV Autopilot* blocks [26]. Continuous *Feedback* from each *Agent Behavior Manager* block, namely the status of the BT and the drone's battery level, is sent back to the planner, so that it can react to certain events. The *Task Outcome* of each task (i.e., success or failure) is also communicated to the *Agent Behavior Manager* by the *Low-Level Controllers*. The chain ends with the control signals generated by the *Low-Level Controller* blocks to make the UAVs fly.

A. High-Level Planner

The *High-Level Planner* is a centralized module of the software architecture. This module is in charge of tasks planning, i.e., it decides which task will be assigned to each UAV. Mission high-level actions (i.e., inspection, monitoring, and delivery) are encoded as human gestures provided by the human crew on the site. Multiple actions could be requested simultaneously. For instance, the crew may ask to inspect a set of target points (i.e., points of interest for the action), performing a visual examination of the power equipment and their surroundings, while the operation of a worker on a nearby power tower is monitored, e.g., providing views with multiple cameras in formation. Recall that there exist multiple UAVs for concurrent mission objectives, and that certain vehicles may play different roles (inspection or monitoring) depending on the needs.

To come up with an action assignment that satisfies the mission objectives, a First-Come-First-Serve scheduling policy is implemented to arrange actions such that maximum priority operations are taken. Besides, actions are endowed with positive numerical weights that can be tuned to parametrize the execution based on the UAV type. Before allocating them to UAVs, actions are arranged within a queue that follows an ascending order based on the weights that have been assigned to each action. The so-formulated scheduling policy allows us to capture user preferences, i.e., the importance or priorities of different actions, and to adapt the mission in case of incompatible tasks or with performance preferences, by changing the actions' position within the queue.

Once an action has been taken (inspection, monitoring, or delivery), a task allocation problem is formulated to decide which task (i.e., points of interest for the action) will be assigned to each UAV. A resource-constrained problem deals with the task assignment, where the resources are the number of available UAVs, their capabilities and remaining battery

levels. Given the complexity of the problem, which may be intractable when the number of tasks, UAVs and mission constraints increases, we propose an heuristic to determine adequate feasible solutions. Specifically, a minimization (1) is carried out over a cost function defined as the sum of three terms:

$$i^{\star} = \arg\min_{V} J_1(V) + J_2(V) + J_3(V); \tag{1}$$

where V refers to the set of available UAVs and $i^{\star}$ represents the optimum tasks sequence for the set of UAVs V. $J_1(V)$ represents a cost per type of UAV, in a way to reward the use of most suitable UAVs to cope with the task execution (inspection and monitoring UAVs can play the same role). $J_2(V)$ is a travel cost, which takes into consideration the distance separating the UAV position to the task start. Last, $J_3(V)$ is an interruption cost, which penalizes interrupting another task in execution to take a new one, according to the action priorities defined by the user. This way, tasks are assigned to UAVs with the lowest cost per task.

This task assignment procedure can be run both when mission planning is conducted offline, and during the execution of the mission, when a replan request may be triggered due to unforeseen or external events. In these situations, the *High-Level Planner* checks whether task reallocation is needed. In particular, replanning is triggered every time a running task is finished, a new task arrives or the parameters of a previously commanded task are modified. Also, replanning is required when a UAV gets disconnected or re-connected due to a communication drop-out or a battery fault occurs, that causes a sudden decrease of the UAV remaining flight time making unfeasible for the vehicle to finish its plan. A watchdog timer helps to manage brief UAV disconnection issues by avoiding replanning operations when not needed.

In order to comply with battery constraints, recharging operations are included as additional tasks in the final plan. Battery consumption is estimated (using a constant rate model) considering the required travel distance for each assigned task. Then, depending on the remaining battery level, recharges are placed either before starting a new task or in the middle of a task, breaking it in two parts, so that the UAV can resume later (e.g., inspecting part of the points in an inspection action, stopping to recharge, and then continue). *Artificial* actions so that a UAV *waits* on its recharging spot can also be scheduled to synchronize task execution in cases of multi-UAV tasks (e.g., if several UAVs need to start a monitoring action together). These actions are in addition to those provided by the problem description.

Figure 3 shows an illustrative example with the plans for 2 UAVs performing a mission considering 4 actions. The human worker supplies the UAV team with a list of actions in the order: delivery a tool, inspect an area of interest, perform a monitoring operation, and then inspect another part. For the considered scenario, only the UAV-1 is supposed to be capable of performing delivery tool operations. Given the limited battery capacity, UAV-2 breaks its *monitoring* operation to recharge in the middle. Meanwhile, UAV-1 takes its role. An artificial *wait* action is scheduled before for task synchronization.

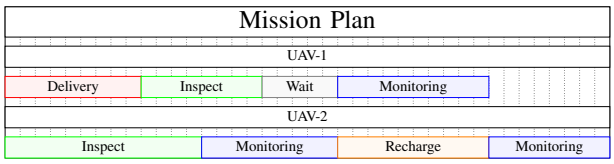


Fig. 3: Example of a mission plan with 2 UAVs and 4 actions. Different colors indicate the type of action, while their duration is represented by the horizontal axis.

After mission planning, the *High-Level Planner* outputs a list of assigned and feasible tasks for each UAV. The tasks are provided as inputs to the N-instances of the *Agent Behavior Manager*, as many as the available vehicles, which are in charge of extracting the tasks' parameters for the corresponding *Low-Level Controllers* where complex behaviors are encoded [14], [23]–[25].

B. Agent Behavior Manager

The *Agent Behavior Manager* runs as a distributed control chain on board each UAV. This module mainly implements a state machine encoded as a BT, which monitors the UAV state and the task outcome and reacts to any possible failure or unexpected events, activating contingency actions and requesting a new plan to the *High-Level Planner*. Such an approach preserves the key properties of the low-level control systems while ensuring high modularity of the whole control structure. It is indeed worth mentioning that BTs unlike FSMs guarantee bidirectional control transfers (up and down the control chain) by replacing the concept of *transitions* with the propagation of a signal through the tree. A complete description of the BT syntax, semantics and functioning can be found in [21], and here it is not reported for the sake of brevity.

In short, let a BT be a directed tree embedding the usual definition of nodes, root, leaves, children, and parents. In a BT each node belongs to one of the following categories: *Fallback*, *Sequence*, *Parallel*, *Action*, and *Condition*. Succeeds, Fails, and Running conditions of these node types are summarized in Table I. Leaf nodes are either *Actions* or *Conditions*, while interior nodes are either *Fallbacks*, *Sequences*, or *Parallels*. Each leaf represents a particular conclusion or action to be carried out, and each nonleaf represents a predicate to be checked. BTs operate propagating a *tick* signal from the root downwards, checking nodes' status according to their operating rules, until it reaches a leaf node, and executing each node's callback in the process.

A hierarchical BT, i.e., the *Main Tree*, encodes the behavior of each UAV, as depicted in Figs. 4, 5 and 6. This *Main Tree* takes as input the list of tasks from the *High-Level Planner*. If the mission is not over, the battery level is checked, and if the UAV has a wrong tool, this is dropped (*Drop Tool Tree*). Depending on the next assigned task, the corresponding subtree (i.e., *Monitoring Task Tree*, *Inspection Task Tree* or *Delivery Tool Task Tree*) is executed, if in *Idle*

NODE TYPE	SUCCEEDS	FAILS	RUNNING
Fallback	If one child succeeds	If all children fail	If one child returns running
Sequence	If all children succeed	If one child fails	If one child returns running
Parallel	If $\alpha \in \mathbb{N}^+$ children succeed	If $\beta > \alpha$ children fail, with $\beta \in \mathbb{N}^+$	Else
Action	Upon completion	When impossible to complete	During completion
Condition	If true	If false	Never

TABLE I: Nodes type of a BT.

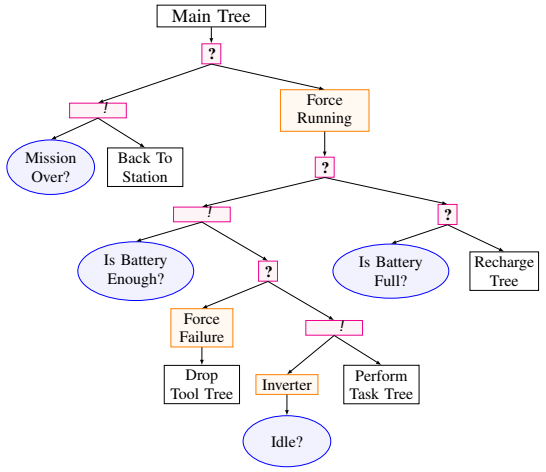


Fig. 4: Main Tree. Reactive control nodes (e.g., *Fallback* or *Sequence*) are shown in magenta, leaf nodes (*Action* and *Condition*) in blue, and decorator nodes (to transform the result of a child) in orange. Normal control nodes and subtrees are in white.

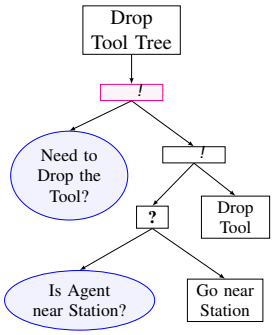
the UAV goes to the recharging station. If a failure is detected (i.e., a communication drop-out or a lack of battery), there is an emergency protocol that empties the UAV's task queue, so that the BT detects no tasks and activates the *Idle* condition to head the recharging station. Meantime, a failure to execute the task is reported to the *High-Level Planner*, who triggers a replanning procedure that redistributes the tasks among the available UAVs to ensure the mission is accomplished successfully.

IV. SIMULATION RESULTS

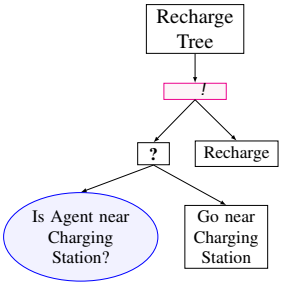
This section showcases the feasibility of the proposed software architecture through several use cases in a realistic simulation setup. In particular, we run the software architecture in a quite close to real application scenario using the Gazebo robotic simulator, exploiting the advantages of Software-In-The-Loop simulations [27]. The software stack was coded using the Robot Operating System (ROS) Melodic Morenia running on Ubuntu 18.04. The *BehaviorTree CPP* library[2] has been used to code the BTs. All simulations were performed on a laptop with an Intel-Core i7-7700 processor (3:60 GHz) and 16GB RAM. Figure 7 depicts a snapshot of the monitoring task, while illustrative videos with the simulations are available at `http://mrs.felk.cvut.cz/bt-planner`. The code has been released as open-source[1] making it possible to replicate the obtained results.

In order to test the interaction of the UAVs with a realistic environment, the simulated scenario includes power towers of 20 m high along with the corresponding wires and a human worker. All Gazebo models and mesh files have also been made available[1]. For the sake of simplicity and ease of experimentation, human gestures to command high-level actions and low-level controllers to perform the tasks have been simulated using console user inputs and idle states, respectively. Note that this does not imply a loss of generality as human gestures and low-level controllers are only input and output interfaces for the proposed software architecture, respectively.

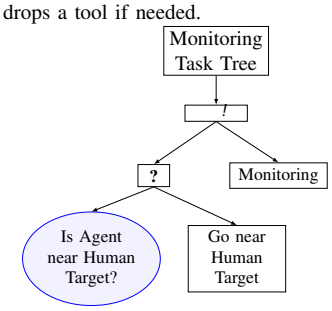
[2] `https://github.com/BehaviorTree/BehaviorTree.CPP`



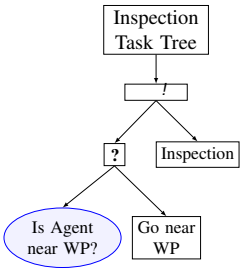
(a) Drop Tool Tree: subtree that drops a tool if needed.



(b) Recharge Tree: subtree describing the recharge operation.



(c) Monitor Tree: subtree dealing with the monitoring operation.



(d) Inspection Tree: subtree detailing an inspection task.

Fig. 5: Subtrees encoding the basic actions for the tasks described in Section II.

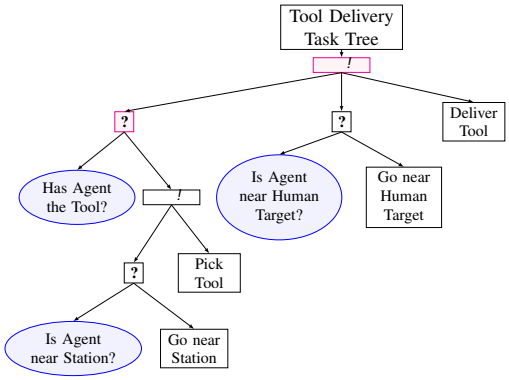


Fig. 6: Delivery Tool Tree: subtree controlling tool delivery tasks.