2022 IEEE 18th International Conference on Automation
Science and Engineering (CASE)
August 20-24, 2022. Mexico City, Mexico

2022 IEEE 18th International Conference on Automation Science and Engineering (CASE) | 978-1-6654-9042-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/CASE49997.2022.9926704

A Reactive Energy-Aware Rendezvous Planning Approach for Multi-Vehicle Teams

Kenny Chour[1], Jean-Paul Reddinger[2], James Dotterweich[2], Marshal Childers[2], James Humann[2], Sivakumar Rathinam[1], Swaroop Darbha[1]

***Abstract*— This paper addresses a type of rendezvous recharging problem involving a team of unmanned aerial and ground vehicles (UAVs, UGVs). UAVs are tasked with long-term surveillance and reactivity to events over a wide area, but are subject to energy constraints limiting their operation. UGVs can support UAVs by replenishing their battery via docking. The presented problem is a generalization of well-known NP-Hard problems in vehicle routing. We present a multi-level framework that decouples the challenging problem into smaller sub-problems for solving. The resulting framework is one that interleaves planning and execution for a team of vehicles, allowing for long-term operation and reactivity. We also present simulation results to validate our proposed approach.**

I. INTRODUCTION

Unmanned vehicles have become widely popular among researchers in the last decade. Teams consisting of unmanned aerial vehicles (UAVs) and ground vehicles (UGVs) can compliment each other in applications such as fire fighting, border security, and infrastructure inspection ([1], [2], [3]). A common theme is *persistent monitoring*, where vehicles must move periodically around areas of a map over a period of time. A challenge with using UAVs is limited energy supply, i.e., the vehicle must be replenished with energy frequently as needed. To support this effort, static and mobile charging stations are often used.

In this work, we address a long-term surveillance problem involving a team of UAVs with limited energy capacity supported by mobile charging UGVs called *rendezvous recharging problem*. A primary objective is to survey nodes along a known graph network continuously with either vehicle type. However, a caveat is that other related missions may also need to be performed at known or unexpected times. This necessitates an unmanned team that is responsive to suddenly changing priorities in a given environment. Additionally, charging is not instantaneous and the UGV can replenish more than one UAV at once. A final consideration is that UAVs have a limited communication range with UGVs and thus cannot receive plans while in flight. These considerations together culminate into a combined problem

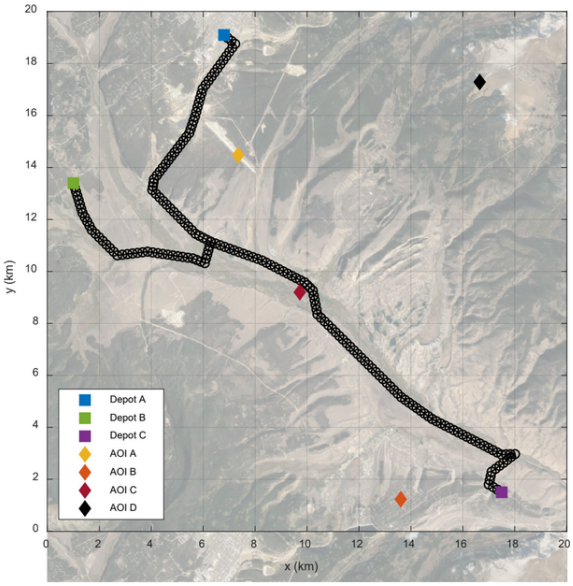

Fig. 1. The 20x20 km real-world site. UGVs are restricted to travel on the black path, while areas of interest (AOI) can only be visited by UAVs. Depots are not considered in this work presently.

of long-term planning and reactivity, which has not been fully addressed in prior work. Aspects of orienteering [4], [5], path coverage [6], and scheduling [7] can also be seen in the problem.

Variants of the rendezvous recharging problem has been well studied in literature with a number of approaches. Kannan et al. [8] introduced the autonomous recharging problem, and proposed an algorithm where worker robots compete to charge under a market-based scheme. Kenzin et al. [9] address the coordinated recharging of heterogeneous mobile robots by devising a job-scheduling algorithm, where robots would rendezvous during specific time windows. In [10], delivery robots are tasked with delivering batteries to task robots, which allow the latter to continue their tasks without rendezvousing. However, none of these approaches directly address our problem of long-term planning and reactivity.

Mixed-integer linear programs (MILPs) are also used to solve some variants related to our problem since constraints can be well expressed. In [11], a MILP formulation of a multi-robot rendezvous problem similar to ours is given and solved. However, the authors assume that the UAV trajectories are fixed and are known ahead of time; their

This material is based upon work supported by the Army Research Laboratory under Cooperative Agreement Number W911NF-21-2-0064.

[1]Department of Mechanical Engineering, Texas A&M University, 3123 TAMU, College Station, TX 77843, USA ckennyc@tamu.edu, srathinam@tamu.edu, dswaroop@tamu.edu

[2]DEVCOM Army Research Laboratory, Aberdeen Proving Ground, MD 21005, USA jean-paul.f.reddinger.civ@army.mil, james.m.dotterweich.civ@army.mil, marshal.a.childers.civ@mail.mil, james.d.humann.civ@army.mil

objective is to obtain optimal paths for the charging robots. We do not make such an assumption on the UAV paths. In [12], an optimal partition decomposition of an environment is presented and an algorithm is developed which attempts to minimize the maximum *age* of targets, i.e. the time elapsed since last visitation (otherwise infinity). We note that the concept of reducing age is related to our approach, but we do not seek an optimal partitioning since dynamic events occur in our environment. Finally in [13], an orienteering problem is generalized with energy replenishment, which is related in spirit to our proposed method. However, the authors there solve an MILP to find an optimal solution, which is not tractable with more number of vehicles and does not address reactivity well. Therefore, we desire a fast and online approach in finding feasible solutions.

To support the previous effort, we propose a multi-level framework that decouples the *rendezvous recharging problem* into a set of smaller sub-problems such as schedule planning, path planning, and execution. The result is an online, energy-aware approach that can be used for long-term operation and is reactive to events in an environment.

II. PROBLEM FORMULATION

Let the UAVs be $\{r_1^a, r_2^a, \ldots, r_M^a\}$ and UGVs be $\{r_1^g, r_2^g, \ldots, r_N^g\}$. We assume the UGV is constrained to travel on a known, obstacle-free roadmap, given as an edge-weighted graph $G = (V, E, W)$. On the other hand, UAVs are not constrained to travel on the same graph and are free to move between any two points in a 2D euclidean space. We assume all UGVs and UAVs are of the same make or type. Define the depth-limited *reachability set* of the UGV as the set of all nodes on the graph network reachable by the UGV up to a certain distance (obtained from Dijkstra's algorithm). Let x_i represent the position of vehicle i. The dynamics of both vehicle types are modeled as a single integrator and are speed limited, where $||\dot{x}_i|| \leq v_{i,lim}$. The UAVs are also subject to limited energy capacity $e_{i,lim}$ and their batteries are assumed to charge and discharge at known rates, which will be expanded upon in the next section. UAVs also cannot be charged beyond maximum capacity. The UGV energy level is not considered in this work.

The UGV's role is to act as a mobile charging station for the UAVs, which the latter must periodically rendezvous with to replenish energy. We assume any UGV r_i^g is able to carry at most $c_{r_i^g}$ number of UAVs at once for charging. An additional constraint for safety reasons is to disallow multiple UAVs from performing a take-off and landing maneuver at the same time from and on a UGV respectively. We assume a nominal or fixed amount of time associated with these maneuvers and denote it by $\Delta_{stagger}$.

The objective for the unmanned team is to cover the known roadmap nodes either through a UGV visitation or UAV flyover. Additionally, high-priority nodes will also need to be visited. Such a scenario is realistic in many military and civilian domains. One simplifying assumption we make is that these sudden changes are made available to the team. However, the UAV is only allowed to receive plans while it is docked with a UGV. Once a UAV has left its UGV, it cannot receive any plan updates from the UGV. However, we allow the UAV to have very limited close-range communication with the UGV via notifying pings. These pings allow the UGV to decide whether to allow the UAV to dock. Additionally, the UAV is allowed some intelligent behavior such as automatically perching when it does not receive a landing command from the UGV after some time.

A. UAV Power Model

The UAVs follow an empirically derived power curve (obtained from the authors in [14]) during discharge, $P = 0.0461||\dot{x}_i^3|| - 0.5834||\dot{x}_i^2|| - 1.876||\dot{x}_i|| + 229.6$ which is only a function of speed. However, when the UAV is perched its power consumption drops to 13 W. When the UAV is charging, the power curve is,

$$P = \begin{cases} 310.8, & e \leq 270.4kJ \\ 19.965(287700 - e), & 270.4kJ < e \leq 287.7kJ \end{cases} \tag{1}$$

Furthermore, UAVs consume 7.2 kJ (30s) and 4 kJ (6s) of energy (time) for any landing and take-off maneuver respectively.

III. METHODOLOGIES

We propose a multi-level framework that decouples the rendezvous problem into separate manageable sub-problems: deployment, scheduling, and path planning. The deployment algorithm is used to coordinate all vehicle behaviors such as UAV landings and takeoffs, as well as UGV stop and go maneuvers. For each r_g, we maintain a list of UAVs currently docked (*chargingList*) and a list of UAVs in the vicinity that are ready to dock (*readyList*). For the coordination to work correctly, a list of UGV path segments ($\mathcal{P}_{r^g}$) is maintained as shown in Fig. 2. A subset of points along the full path are key points where the UGV must either stop or go for both UAV takeoff and rendezvous activites. Together with *clockTime*, these points form a trajectory-like list and and are called *milestones*. Let the set of all milestones for a given UGV be $\mathcal{M}_{r^g}$. Fig. 2 exemplifies the gradual addition of milestones to an initially empty $\mathcal{M}_{r^g}$. We attempt to reuse as much of the existing paths in $\mathcal{P}_{r^g}$ as possible to avoid replanning. However, $\mathcal{P}_{r^g}$ can be extended with a new path segment as can be seen in the middle figure in Fig. 2; this will only occur when milestones are appended to a $\mathcal{M}_{r^g}$.

Each time a milestone is to be added to $\mathcal{M}_{r^g}$, we have to ensure the UGV can actually reach every milestone on time; this can be done using a scheduler that explicitly considers conflicts. The *clockTime* of a *milestone* is determined by a scheduler which maintains a set of conflict-free time intervals called *appointments*, $\mathcal{A}$. These appointments are created using a set of interval trees for conflict detection, $\mathcal{C}$ (see Sec. III-B) and represents the duration in which a UAV is docked with a UGV. Additionally, there are starting and ending delays ($= \Delta_{stagger}$) within an appointment. The UGV must remain stationary during the starting delay

interval. The location x of a *milestone* is determined by the UGV's path planning module. For this work, each time a UAV rendezvous with the UGV (r^g), two new *milestones* (takeoff and rendezvous) are added to $\mathcal{P}_{r^g}$ without violating the reachability of pre-existing milestones.

A. Deployment

At initialization, we assume all UAVs are sufficiently charged. For each UGV r^g, we first schedule the takeoff and rendezvous of every UAV in the *chargingList*. These takeoffs are spaced apart by $\Delta_{stagger}$. Then, the rendezvous of the UAVs in the *readyList* are scheduled with *appointments* in batches of size c_{r^g}. The positions of the *milestones* are then planned for every scheduled *appointment*. The rendezvous positions for the UAVs in the *readyList* are fixed to the UGV's initial position, as well as the takeoff positions for all but the last c_{r^g} (or 1) UAVs. This ensures non-docked UAVs in the *readyList* are not abandoned by the UGV. This process presents us with an initial $\mathcal{C}$, $\mathcal{M}_{r^g}$, $\mathcal{P}_{r^g}$, and $\mathcal{P}_{r^a}$ for all UGVs and UAVs.

Algorithm 1: Deployment

```
1  Input: Initialized C, M_{r^g}, P_{r^g}, and P_{r^a} for all
     UGVs and UAVs
2  while True do
3      foreach r^g ∈ r^g_1, ..., r^g_m do
4          m ← M_{r^g}.milestones[0]
5          if r^g is close to m then
6              if m.isTakeoff == True ⋀ clock
                 >= m.clockTime then
7                  Upload m.r^a plans
8                  Delete m from M_{r^g} and adjust path
                     accordingly
9              else if m.isRendez == True ⋀ clock
                 >= m.clockTime then
10                 Make r^g brake for Δ_stagger
11                 if m.r^a ∈ readyList then
12                     r^g sends land command to m.r^a
13                     chargingList.append(r^a)
14                     Delete m from M_{r^g} and adjust
                         path accordingly
15                     (P_{r^g}, P_{r^a}) ←
16                     planTakeOffRendez(C, r^a, r^g, M_{r^g}, P_{r^g})
17             else
18                 Make r^g brake // Makes r^g wait
                       till clock == m.clockTime
19         else
20             Send next path segment of P_{r^g} to r_g if
                 awaiting mission
21             Send go command to r_g
```

The main loop ensures the unmanned team will be operating continuously. We assume a clock signal *clock* is available, representing wall time. For each UGV r^g, we check to see if it is close enough to the next milestone (Line 5). If the *clock* is lesser than $m.clockTime$, then the UGV will pause (Line 18). Then, if the milestone m corresponds to a takeoff and the clock time exceeds the milestone clock time, the UAV flight plans are uploaded for immediate takeoff (Line 6-7). Consequently, m is deleted from $\mathcal{M}_{r^g}$ (Line 8).

If m is a rendezvous type (line 9) and clock time is satisfied, then the ready UAV is commanded to land (line 12). The UAV is then added to the *chargingList*, and a new takeoff and rendezvous milestone is generated for $\mathcal{M}_{r^g}$. In line 15, the updated UGV and UAV path is produced. If the UGV has finished following a path segment and is currently awaiting (idle), the next path segment is sent (Line 20). Finally if the UGV is not near the next milestone, it is commanded to do waypoint following to reach it (line 21-22).

B. Conflict-Based Interval Trees

Greedy scheduling is used to generate conflict-free appointments or $\mathcal{A}$ for the UAVs. The scheduler (Sec. III-C) is called each time a UAV successfully docks in order to generate a predicted appointment for the next rendezvous. We define $\mathcal{A}$ to have the following attributes: i) *startTime* is the starting time of the appointment, set to the arrival of a UAV prior to docking; ii) *server* is the associated charging pad that services a UAV with energy; iii) *client* represents the unique name of the UAV; iv) *startDelayInt*, *serviceInt*, *endDelayInt* are all time intervals that make up a single appointment and are respectively the service staring delay, actual servicing, and service end delay intervals; Note that the left endpoint of each interval can be expressed as a function of the start time.

Generating a feasible $\mathcal{A}$ requires other appointments that may overlap with it. To efficiently obtain such overlaps or *conflicts*, we rely on a data structure called an *interval tree*. Such trees allow for the insertion, deletion, and querying of a time interval (for overlap) in $\mathcal{O}(logN)$, where N is the number of intervals maintained in the tree [15], [16]. There are three types of conflicts we consider: *delay-based*, *server-based*, and *reachability-based*. *Delay-based* conflicts occur when UAVs attempt to takeoff from or land on a UGV at the same time; *Server-based* conflicts occur when more than one UAV attempts to dock with the UGV at the same charging pad; *Reachability-based* conflicts occur when the scheduled *appointment* affects the reachability of future milestones in the UGV's path. We maintain interval trees for each type of conflict. More specifically, there is an interval tree for each charging pad for handling server-based conflicts, while there are two individual interval trees for handling delay-based and reachability-based conflicts. In total, there are $c_{r^g}+2$ interval trees used. We call this approach *Conflict-based Interval Trees* and denote the collection of all trees by $\mathcal{C}$.

Alg. 2 shows how a feasible $\mathcal{A}$ can be generated. The algorithm takes an initial $\mathcal{A}$ and repeatedly modifies its starting time until no more conflicts exist. This is done without ever changing the exact duration of each interval in $\mathcal{A}$; only the starting time is modified. We also maintain two lists, *conflictList* and *serverConflictList* to differentiate between conflict types. After (re-)initializing in Lines 2-5,

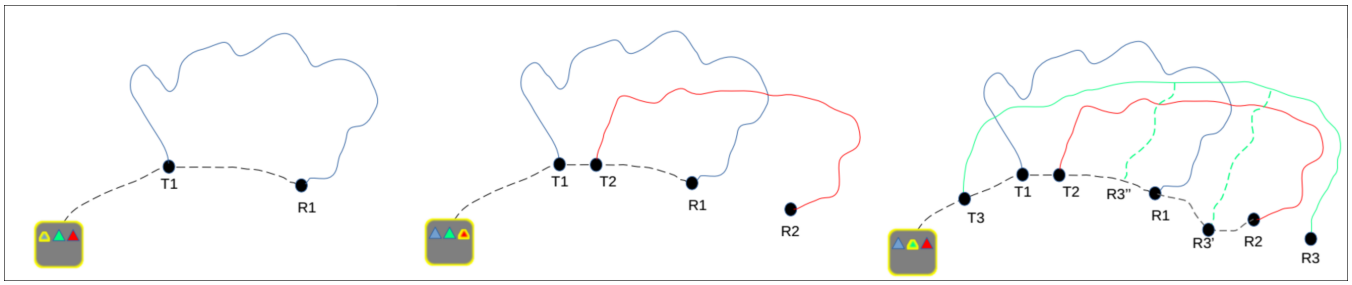

Fig. 2. Dynamic update of UGV (lower left box) path as UAVs are processed in sequence. **Left**: the path (dashed) connecting T1 and R1 nodes is appended. **Middle**: T2 is inserted, but R2 is appended. **Right**: T3 is inserted, but there are many choices for R3.

we attempt to store any server-based conflicts and a conflict-free server. Later in Lines 11-13, we store any delay and reachability-based conflicts. In Line 14, if no conflicts exist we update the server and delay-based interval trees with $\mathcal{A}$. Otherwise, $\mathcal{A}$'s start time is adjusted based on the earliest finish time or *makespan* over all conflicts.

Algorithm 2: addToCBITree($\mathcal{C}, \mathcal{A}$)

```
1  while True do
2      conflictList = []
3      serverConflictList = []
4      bestServer ← NULL
5      foreach t ∈ {serverIT_1, ... serverIT_{c_rg}} do
6          if t.query(A.serviceInt) ≠ ∅ then
7              serverConflictList.append(
8              t.query(A.serviceInt))
9          else
10             bestServer ← t.server
11     conflictList.append(delayIT.query(A.startDelayInt))
12     conflictList.append(delayIT.query(A.endDelayInt))
13     conflictList.append(reachIT.query(A.fullInt))
14     if conflictList ∪ serverConflictList = ∅ then
15         A.server ← bestServer
16         bestServer.serverIT.insert(A.serviceInt, A)
17         delayIT.insert(A.startDelayInt, A)
18         delayIT.insert(A.endDelayInt, A)
19         return A
20     else
21         A.startTime ←makespan(conflictList∪
22         serverConflictList)
```

C. 2-Phase Scheduling

The actual scheduling process is a 2-phase procedure involving *correction* and *prediction* steps for an appointment; both steps occur whenever a UAV has docked. First, we assume a prediction (appointment) initially exists as a result of initialization steps during deployment step. During the correction step in Alg. 3, a UAV's actual arrival time and detected energy level upon docking is used to correct any inaccuracies in prior predictions (Lines 1-2). Here the predicted service interval may be extended as much as possible to maximize charging, but note that the correction cannot violate any existing appointments. A new takeoff milestone is created with NULL position and time set to the right end point of $\mathcal{A}$'s service interval (Line 4). After a sort on $\mathcal{M}_{r^g}$, the new milestone is returned (Line 5-6). During the prediction step in Alg. 4, a new $\mathcal{A}$ is created and modified using the *addToCBITree* routine (Lines 1-3). A new rendezvous milestone is added to $\mathcal{M}_{r^g}$ without a position x specified (Lines 4-5) and is also returned.

Algorithm 3: scheduleCorrection($\mathcal{M}_{r^g}, r^a, t_c$)

```
1  A ← Get most recent predicted appointment of r^a
2  Update A // correct predicted
       appointment with new start time t_c
       and service end time upon arrival
       and landing.
3  A.predicted ← False
4  Add a takeoff milestone m to M_{r^g}, with m.x =
     NULL and m.clockTime = A.serviceInt.right
5  Sort M_{r^g} by clockTimes
6  return m
```

Algorithm 4: schedulePrediction($\mathcal{C}, \mathcal{M}_{r^g}, r^a, t_c$)

```
1  Create A using takeoff time, estimated energy at
     takeoff, desired flight time, and delay times
     calculated using t_c
2  A.predicted ← True
3  A ← addToCBITree(C, A)
4  Add a rendezvous milestone m to M_{r^g}, with m.x =
     NULL and m.clockTime = A.startTime
5  Sort M_{r^g} by clockTimes
6  return m
```

D. Milestone Path Planning

Once scheduling is performed, we have a time-based ordering of events (milestones) in $\mathcal{M}_{r^g}$ and must begin to define their positions. In Alg. 5, two new milestones T_{new} and R_{new} are generated without positions specified (Lines 2-3) for a given UAV upon docking. Their indices are given in Lines 4-5.

Algorithm 5: planTakeoffRendez($\mathcal{C}, r^a, r^g, \mathcal{P}_{r^g}, \mathcal{P}_{r^g}$)

1 $t_c \leftarrow$ current time
2 $T_{new} \leftarrow$ scheduleCorrection($\mathcal{M}_{r^g}, r^a, t_c$)
3 $R_{new} \leftarrow$ schedulePrediction($\mathcal{C}$,M_{r^g},r^a, t_c)
4 $T_{index} \leftarrow \mathcal{M}_{r^g}.index(T_{new})$
5 $R_{index} \leftarrow \mathcal{M}_{r^g}.index(R_{new})$
// Takeoff and rendezvous positions sampled based on a UGV's path segments
6 $S \leftarrow \emptyset$ **while** $|S| < s_{lim}$ **do**
7 **if** $Rindex < len(\mathcal{M}_{r^g})$ **then**
8 $s \leftarrow$ Sample a takeoff point from $\mathcal{P}_{r^g}$ and a rendezvous point from the UGV's reachability set such that existing milestones are not impacted
9 **else if** $Tindex > len(\mathcal{M}_{r^g}.milestones)$ **then**
10 $s \leftarrow$ Sample both takeoff and rendezvous point from UGV's reachability set such that existing milestones are not impacted
11 **else**
12 $s \leftarrow$ Sample a takeoff point and rendezvous point from $\mathcal{P}_{r^g}$ such that existing milestones are not impacted
13 $S \leftarrow S \cup \{s\}$
14 $s^*, \mathcal{P}_{r^a} \leftarrow \arg\max_{s \in S} RewardFunc(s)$
15 Modify $\mathcal{P}_{r^g}$, $\mathcal{M}_{r^g}$ with s^*
16 Update reachIT
17 **return** ($\mathcal{P}_{r^g}$, $\mathcal{P}_{r^a}$)

A sampling-based approach is used to obtain s_{lim} number of samples (Line 6) for the pair (T_{new}, R_{new}). The rest of the algorithm (Lines 7-12) is simply logic that details the different ways in which a takeoff or rendezvous point can be chosen from the given UGV path $\mathcal{P}_{r^g}$. After a desired number of samples is obtained (Line 13), the path with the greatest reward is constructed using Traveling Salesman Problem heuristics [17] (Line 14); the reward function is simply the sum of all rewards along a path divided by path length. Then, the $\mathcal{M}_{r^g}$ and $\mathcal{P}_{r^g}$ is modified with the best sample pair (Line 15), while reachIT is updated based on reachability slack (Line 16), before return updated paths (Line 17).

IV. SIMULATION RESULTS

The proposed multi-layer approach was implemented in a multi-agent simulator written in Python. Solutions were computed on a laptop computer running an Ubuntu 18.04 operating system with an Intel i7-7700HQ processor and 16GB of RAM. The simulation incorporates both power dynamics from section II-A and a single integrator model for each vehicle. The road network considered in this simulation is shown in Figure 1, which is based on data from a 20 square kilometer real-world site. A problem instance consists of 3 UAVs and 1 UGV, and a random starting location. Two of the UAVs are assumed to be docked with the UGV, while the remaining UAV is perched nearby. If any of the UAVs run out of power or attempts to land on a non-available charging pad, an error is thrown. For all of our instances, we did not see any issues.

TABLE I
AMOUNT OF TIME SPENT CHARGING

Time Spent Charging (hr)				
UAV1	UAV2	UAV3	Pad1	Pad2
19.73	18.65	17.44	33.55	22.28

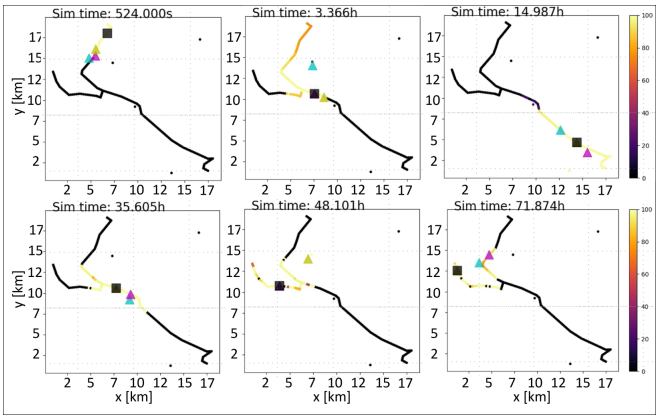

Fig. 3. Team position over time for a 72-hour simulation (UGV is the square, UAVs are triangles). The youth value is at most 100 for the most recently visited nodes and 0 for the oldest nodes.

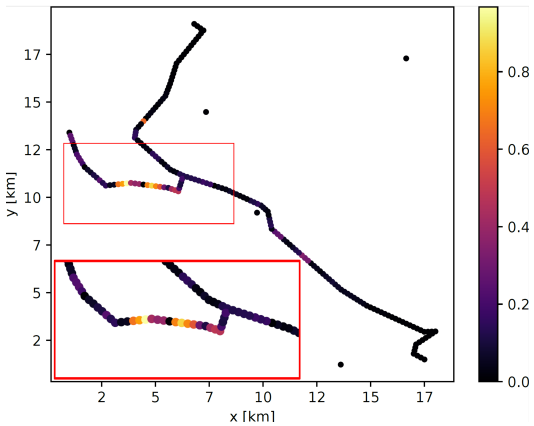

Fig. 4. Heatmap representing each node's total age, i.e. the total accumulated time visited by any vehicle. The boxed region is where the team spent the most time.

We first investigated the team's ability to navigate around the map. This is done by exploiting reward-based TSP heuristics in Alg. 5. We set each node to have a maximum reward value of 10, while AOIs have a reward of 1000. The reward value is reset to zero upon visitation and gradually grows to its max value over time. This reward system biases milestone selection over areas with higher rewards, causing the team to move towards them. We also define the youth of any node on the graph network as,

$$youth_v = \max(\min(100 - e^{d\tau_v/b}, 100), 0), \text{ for } v \in V. \quad (2)$$

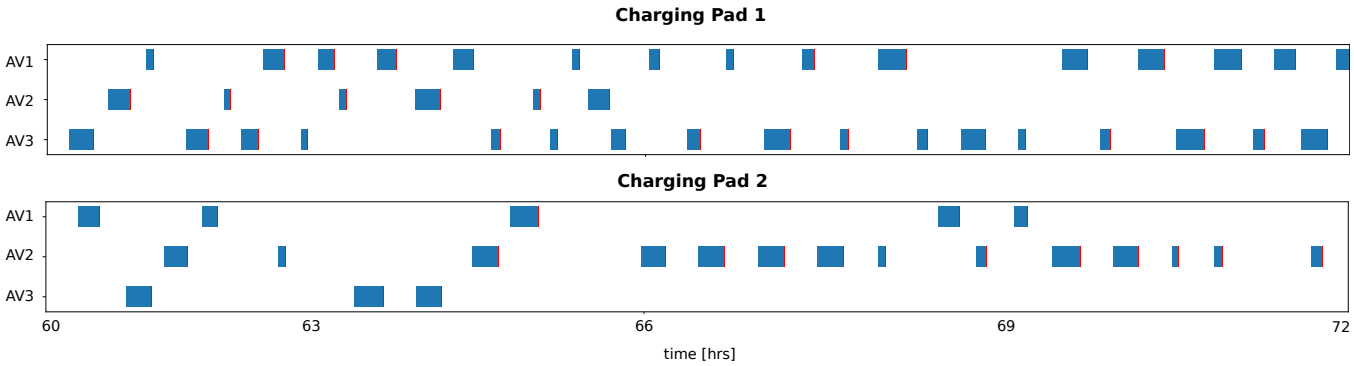

Fig. 5. 2 feasible schedules presented for the team of 3 UAVs and 1 UGV. The two subfigures represent the charging schedule (in blue) for the UGV's charging pad 1 and 2. There is no queuing time due to the absence of overlapping time intervals within a charging pad.

The youth value is a unitless quantity, representing the most recently visited nodes, shown in yellow in Fig. 3. Clearly, the team is able to visit different areas of the map over the entire simulation period, proving the framework can work.

Next, we investigated the *age* of each node, which represents the total visiting time by any vehicle, accumulated over the simulation period. This quantity is depicted as a heatmap in Fig. 4. We see that most of the time is spent in the boxed region of the map, though other parts of the region have also been visited. We posit that the reward system employed is most likely leading to a locally optimal area, which prevents the team from moving away. Future work will investigate better biasing techniques to ensure a more robust coverage technique.

A feasible schedule is shown in Fig. 5 for the final 12-hour period over the entire simulation period, where each interval (blue) indicates the times during which a UAV is charging. Since no overlap is shown, there is no queuing time necessary. The charging time per vehicle and server (charging pad) is also shown in Table I, indicating a nearly uniform amount of charging time per UAV.

V. CONCLUSIONS

In this work, a multi-level coordination, scheduling, and planning algorithm for a rendezvous recharging problem was presented. Simulation results for a team of 1 UGV and 3 UAVs over a 72 hour period show the effectiveness of our approach. Though our results are not optimal, the planner is highly efficient compared with MILP approaches. Future work will look towards decentralizing the algorithm and scaling for more vehicles. Future work will also look into longer range UAVs, and more sophisticated algorithms.

REFERENCES

[1] A. Ollero and L. Merino, "Unmanned aerial vehicles as tools for forest-fire fighting," *Forest Ecology and Management*, vol. 234, no. 1, p. S263, 2006.

[2] M. Banić, A. Miltenović, M. Pavlović, and I. Ćirić, "Intelligent machine vision based railway infrastructure inspection and monitoring using uav," *Facta Universitatis, Series: Mechanical Engineering*, vol. 17, no. 3, pp. 357–364, 2019.

[3] S. Berrahal, J.-H. Kim, S. Rekhis, N. Boudriga, D. Wilkins, and J. Acevedo, "Border surveillance monitoring using quadcopter uav-aided wireless sensor networks," *Journal of Communications Software and Systems*, vol. 12, no. 1, pp. 67–82, 2016.

[4] W. Souffriau, P. Vansteenwegen, G. Vanden Berghe, and D. Van Oudheusden, "The Multiconstraint Team Orienteering Problem with Multiple Time Windows," *Transportation Science*, vol. 47, no. 1, pp. 53–63, Feb. 2013. [Online]. Available: http://pubsonline.informs.org/doi/abs/10.1287/trsc.1110.0377

[5] A. Gunawan, H. C. Lau, and P. Vansteenwegen, "Orienteering Problem: A survey of recent variants, solution approaches and applications," *European Journal of Operational Research*, vol. 255, no. 2, pp. 315–332, 2016, publisher: Elsevier B.V.

[6] Y. Chen, H. Zhang, and M. Xu, "The coverage problem in UAV network: A survey," *5th International Conference on Computing Communication and Networking Technologies, ICCCNT 2014*, no. 3, pp. 3–7, 2014, iSBN: 9781479926961.

[7] J. D. Ullman, "Np-complete scheduling problems," *Journal of Computer and System sciences*, vol. 10, no. 3, pp. 384–393, 1975.

[8] B. Kannan, V. Marmol, J. Bourne, and M. B. Dias, "The Autonomous Recharging Problem: Formulation and a market-based solution," in *2013 IEEE International Conference on Robotics and Automation*. Karlsruhe, Germany: IEEE, May 2013, pp. 3503–3510. [Online]. Available: http://ieeexplore.ieee.org/document/6631067/

[9] M. Kenzin, I. Bychkov, and N. Maksimkin, "Coordinated Recharging of Heterogeneous Mobile Robot Teams during Continuous Large Scale Missions *," in *2020 7th International Conference on Control, Decision and Information Technologies (CoDIT)*. Prague, Czech Republic: IEEE, June 2020, pp. 745–750. [Online]. Available: https://ieeexplore.ieee.org/document/9263974/

[10] N. Kamra, T. K. S. Kumar, and N. Ayanian, "Combinatorial Problems in Multirobot Battery Exchange Systems," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 2, pp. 852–862, Apr. 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8125731/

[11] N. Mathew, S. L. Smith, and S. L. Waslander, "Multirobot Rendezvous Planning for Recharging in Persistent Tasks," *IEEE Transactions on Robotics*, vol. 31, no. 1, pp. 128–142, Feb. 2015. [Online]. Available: http://ieeexplore.ieee.org/document/7001257/

[12] S. Seyedi, Y. Yazicioglu, and D. Aksaray, "Persistent Surveillance With Energy-Constrained UAVs and Mobile Charging Stations," *arXiv:1908.05727 [cs, eess]*, Aug. 2019, arXiv: 1908.05727. [Online]. Available: http://arxiv.org/abs/1908.05727

[13] N. D. Wallace, H. Kong, A. J. Hill, and S. Sukkarieh, "The orienteering problem with replenishment," in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2020, pp. 973–978.

[14] A. Hurwitz, J. M. Dotterweich, and T. Rocks, "Mobile robot battery life estimation: battery energy use of an unmanned ground vehicle," in *Energy Harvesting and Storage: Materials, Devices, and Applications XI*, vol. 11722. International Society for Optics and Photonics, 2021, p. 117220I.

[15] R. Sedgewick and K. Wayne, "Algorithms (4th edn)," 2011.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[17] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, II, "An analysis of several heuristics for the traveling salesman problem," *SIAM journal on computing*, vol. 6, no. 3, pp. 563–581, 1977.