2013 IEEE RO-MAN: The 22nd IEEE International Symposium on
Robot and Human Interactive Communication
Gyeongju, Korea, August 26-29, 2013

A Hierarchical Flight Planner for Sensor-Driven UAV Missions

Spencer Clark and Michael A. Goodrich

***Abstract*— Unmanned Aerial Vehicles (UAVs) are increasingly becoming economical platforms for carrying a variety of sensors. Building flight plans that place sensors properly, temporally and spatially, is difficult. The goal of sensor-driven planning is to automatically generate flight plans based on desired sensor placement and temporal constraints. We present a hierarchical sensor-driven flight planning system capable of generating 2D flights that satisfy desired sensor placement and complex timing and dependency constraints. The system makes use of several well-known planning algorithms and includes a user interface. Results demonstrate that the algorithm is general enough for use by a human in several simulated wilderness search and rescue scenarios.**

I. Introduction

Unmanned aerial vehicles (UAVs) are frequently used primarily as sensor platforms, meaning that the UAV is used to carry a sensor to a particular location at a particular time. UAVs are capable of carrying a wide variety of sensors, including the following: cameras (visible [1], [2] and other spectrums [3]), radio antennas, laser range finders [4], radars [5], and radiation [6] and chemical [7] detectors.

In order to accomplish a task using UAV-mounted sensors, the UAV's flight path must place the relevant sensors effectively both temporally and spatially. A camera, for example, should be positioned so that targets are visible within the frame. Often, this is accomplished by a human operator flying remotely or by an autopilot flying a series of waypoints, although there exist algorithms for placing specific sensors at specific locations for specific applications [8]–[10]. Planning (or remotely flying) a path that places sensors effectively isn't easy. This means that, often, the operator must imagine himself or herself in the position of the plane and then take into account a variety of factors including sensor properties (field of view, useful range) and UAV kinematics (airspeed, turning radius).

Current algorithms are either difficult, error-prone, require a high level of attention from operators or they are specific to particular applications and particular sensors. We propose a general approach to planning that is based on a belief that users with minimal training can be empowered to easily create sophisticated flight plans by expressing their high-level sensor goals and constraints to a planning program (see Figure 1). We call this process *sensor-driven planning*. This paper presents both a user interface for specifying flight goals and constraints as well as a general-purpose algorithm for generating flight plans that accomplish the goals while satisfying the constraints.

II. Problem Formulation / Taxonomy

Since the set of possible UAV flight plans is large and we want users to be able to express any desired flight to the planner, it's necessary to come up with a small set of simple building blocks that can describe a wide range of useful flight plans. In this section, we present a taxonomy of sensors and tasks that serve as the basis for the general purpose planner. This taxonomy is not only useful for specifying design requirements, it also enables us to create a hierarchical planner that uses task- and sensor-specific algorithms as needed by the problem.

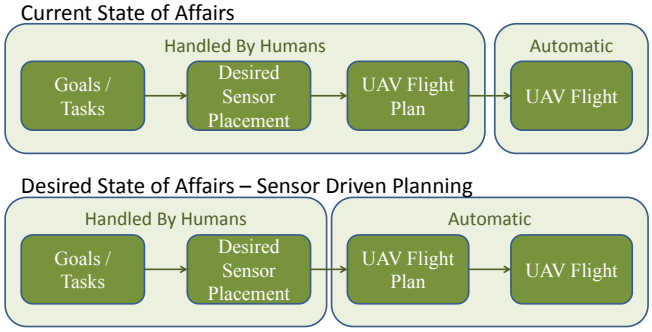

Fig. 1. Currently, the bulk of the UAV flight planning process is handled by humans. We propose that sensor-driven planning enables users to focus on sensor-based flight goals while letting the computer plan the flight.

As discussed previously, UAVs can carry many types of sensor. Fortunately, we can classify them into more general *canonical sensors* based on how a UAV must fly in order to use them effectively. Broadly speaking, a sensor is either directional (like a camera) or omni-directional (like many antennas). Directional sensors need to be pointed at their target whereas omni-directional sensors just need to be within range of it. Both sensor types are parameterized by a maximum usable distance, operationally defined as the maximum distance at which flight tasks can be accomplished with the sensor. Additionally, directional sensors are characterized by a "sensing volume" — for cameras, this is known as field of view. For the purposes of UAV flight planning, we can use these simple characteristics to represent almost any sensor. The planner doesn't need to know anything about the sensors beyond their directionality, maximum range, and (if applicable) sensing volume.

There are a huge number of tasks that a user might want to accomplish with their UAV's sensors. We can't enumerate all of them in a user interface, so instead we propose a set of *canonical tasks* that can act as the building blocks for more complicated tasks. The canonical tasks are *coverage* and *sampling*.[1] The coverage task covers or senses an entire area using a sensor. Aerial photography as in [11] is an

[1]Note that for convenience we have added a third task, *fly through*, in our implementation. This task simply requires that the UAV fly within its area at least once. The addition of this task does not violate our taxonomy as it is really just a degenerate coverage or sampling task.

978-1-4799-0509-6/13/$31.00 ©2013 IEEE

	Directional	Omni-directional
Coverage	Aerial photography	Signals intelligence (SIGINT)
Sampling	Pipeline construction monitoring	Meteorology

TABLE I
EXAMPLES OF THE COMBINATIONS OF CANONICAL TASKS AND SENSORS.

example of a coverage task. The sampling task uses a sensor to gather a number of samples from an area as in [12]. The set of canonical sensors crossed with the set of canonial tasks provides four fundamental sensing options (see table I). Although these tasks seem simple, it is possible to represent a huge spectrum of UAV sensor tasks using the canonical tasks (and a few scheduling parameters) as building blocks.

The canonical tasks can be configured with two different types of scheduling constraints: dependencies on other tasks and valid time windows. When a task α has a dependency on task β it means that α must not be performed until β has been performed. Valid time windows are intervals of time specified by a starting time and an ending time with the ending time strictly greater than the starting time. A flight task must be configured with one or more valid time windows which specify times during which the task may be flown.

There is one final building block in our taxonomy: areas, including task areas and no-fly (obstacle) areas. Task areas should be defined on areas of interest by the operator. One or more canonical tasks can then be assigned to the area. No-fly zones are obstacles that should be avoided by the UAV. For simplicity, we consider areas to be two-dimensional polygons, but future work should extend them to three-dimensional volumes.

In summary, the taxonomy consists of canonical tasks, canonical sensors, timing constraints, and areas. Later, we will discuss how our planner generates flights for planning problems defined in terms of these building blocks, which can represent a variety of UAV missions.[2]

III. COMPLEXITY / PRIOR WORK

Generating a flight that satisfies all tasks and constraints given by a planning problem is difficult. Consider, for example, the problem posed by a flight with n areas, each with a corresponding task to be completed. Generating a flight that is optimal with respect to flight length and that handles all n areas (and their tasks) has essentially solved the traveling salesman problem by deciding the order in which to visit the tasks [10]. This is evidence that, in general, planning optimal solutions to sensor-driven flight problems is very challenging.

Although path planning is a well-studied area, most work focuses on variations of the problem of getting from a starting point to an end point efficiently. Approaches for UAV planning include variants of A* [13], evolutionary algorithms [14], Dubins Curves [15], energy minimization [16], and RRTs [17]. Some algorithms support obstacle avoidance or kinematic constraints. A few plan flights based on sensors. However, algorithms for planning flights with general goals for multiple types of sensors do not seem to exist.

[2]Although the taxonomy can express many UAV missions, there are limitations. It is not a good model for missions with task areas of non-uniform importance, for example.

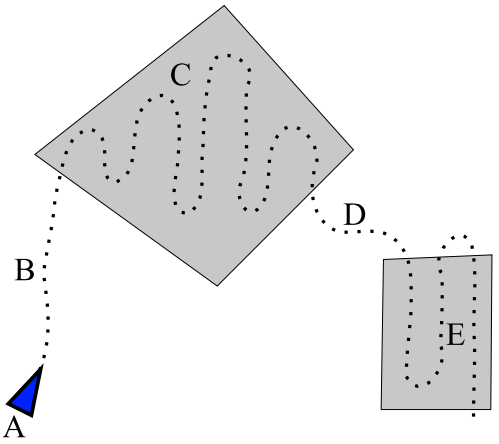

Fig. 2. Each part of the hierarchical planner plays a different role in generating the overall flight. In this example, C and E are sub-flights generated by the sub-flight planner to fly tasks in the shaded areas. B and D are intermediate or connecting flights generated by the intermediate planner, which runs as a subroutine as the scheduler. The scheduler is responsible for deciding when to schedule sub flights. A is the UAV's starting position.

IV. HIERARCHICAL PLANNER

The idea behind hierarchical planning is to break the planning problem into several stages that are more tractable than the overall problem. The taxonomy suggests that many problems can be decomposed into two components: planning a sensor-appropriate flight path for an area, and scheduling flight paths to satisfy timing constraints. Therefore, our approach works by breaking the problem up and planning flights for each flight task in isolation and then scheduling the generated flights into an overall solution. Figure 2 shows how the components of the hierarchical planner generate different pieces of a flight.

Broken up in this manner, our flight planning problem closely resembles scheduling processes on a computer. Flight tasks can be thought of as processes and the UAV can be thought of as a CPU which runs them. Just as a running process on a CPU can be preempted, our planning problems sometimes require a flight task to be interrupted before completion. Just as preempting processes on a CPU has overhead, interrupting flight tasks to handle another has the cost of travel between tasks.

The hierarchical approach has some tradeoffs and limitations. It makes the problem tractable and the algorithm generalizable, but it sacrifices optimality in several places to do so. One limitation with the hierarchical approach and the CPU scheduling metaphor is that the UAV cannot fly more than one task at a time. Thus, we sacrifice optimality when flight tasks are co-located.

At a high level, the hierarchical planner follows the steps given in Figure 3. These high-level steps are easy to understand but the details are more involved. The next sections will discuss the steps in more detail.

1) For each flight task:
   a) Select a task-specific starting position and orientation on or near the task area's boundary.
   b) Generate a "sub-flight" that accomplishes the flight task with its task constraints independent of all other flight tasks. This sub-flight must start at the previously-generated starting position and pose.
2) Generate a schedule that determines when to fly each sub-flight.
3) Using the schedule, build an overall flight by stringing the sub-flights together with intermediate flights.

Fig. 3. The high-level steps of the hierarchical planner.

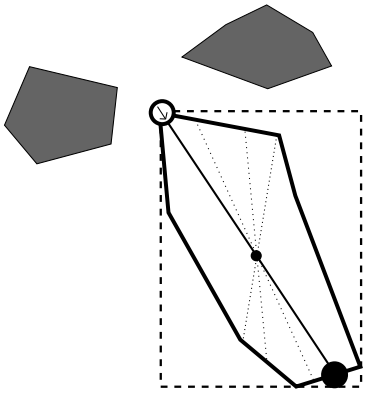

Fig. 4. To choose a starting configuration, a bounding box is calculated around the area. Line segments intersecting the center of the box are calculated at one degree intervals. The end points of the line with the most length within the area's polygon are candidates for the task area's starting position. The candidate point which is closest to the average of all task area's bounding box centers is selected. The orientation points along the line.

A. Task Start Position and Pose

Before the planner can generate sub-flights for each of the flight tasks, positions and orientations from which to start each sub-flight must be chosen. Choosing a good starting position and pose is important and non-trivial. Choosing the optimal starting position and pose (those that will result in the shortest satisfying flight) requires knowing where the UAV will be coming from. Since the schedule isn't computed until after all sub-flights are generated, it is not possible to choose the optimal starting position and orientation. We can however, use a heuristic to choose a starting position and orientation that are likely to give good results.

The goal of this heuristic is to choose a starting position and orientation that are close to other tasks and are likely to allow the UAV to fly for a while without turning. We calculate lines spaced one degree of angle apart passing through the center of the task area's bounding box. The line with the greatest distance within the task area is selected. Next we examine the two end points of the line segment within the task area. The end point which is closest to the average of all task areas' centers is chosen as the starting position. This heuristic seeks to minimize the expected distance between any preceeding tasks and the starting point. The starting orientation is chosen to point towards the center of the task area's bounding box. This process is pictured in Figure 4.

B. An Example Sub-Flight Planner

The sub-flight planner is the part of the hierarchical planner responsible for planning the portions of the overall flight that accomplish tasks. The sub-flights must start in their start position and orientation, accomplish their tasks, and satisfy all task constraints.

Our prototype implementation uses a greedy tree search. The greedy sub-flight algorithm searches a tree with nodes defined by a position and a pose. The root node, for example, is the task area's starting position and pose. The search generates child nodes by making valid moves (according to the UAV's kinematic constraints) from the best current node. Each task type (sampling, coverage, etc.) defines the reward function that rates prospective sub-flights, which are represented by the position and poses of the nodes from root to leaf.

At each iteration the node with the highest reward is removed from a work list. New orientation vectors are calculated based on the current node's orientation and the UAV's kinematic parameters. In addition, new positions are generated by translating the current node's position by each of the previously generated angles. New nodes are created as children of the current node with the generated orientations and positions. Score values are generated for the new nodes using the flight task's reward function and the new nodes are inserted into the work list. This process is summarized in Algorithm 1. Summaries of the reward functions used in the greedy planner for each of the canonical tasks is given in Table II.

Generally, greedy search seems to be a good choice. In our testing it generated satisfactory flights for coverage and sampling tasks very quickly (see Section V). However, its performance is fundamentally tied to the reward functions it is optimizing for (see Table II). The reward functions should be adapted as necessary. One limitation of the coverage task reward function described in Table II is that it assumes directional sensors have sensing volumes with approximately square bases. The best flight for a camera with a very horizontal aspect ratio, for example, may be different from flights for other sensors.

Greedy search is, of course, not the only possible implementation of a sub-flight planner. A* could be considered, but it is not clear how to construct an admissible heuristic for planning the sub-flights, as they are not simple flights from point A to point B. Rapidly-Exploring Random Trees perform inconsistently and would require the addition of another dimension (time) to the problem in order to plan sub-flights that self-intersect or have loop-like shapes, such as sub-flights for coverage tasks. Genetic algorithms were attempted but had difficulty coping with coverage tasks.

C. Scheduler

Once a sub-flight has been generated for each flight task, a scheduler must decide when to fly each one and how to string them together. Scheduling is not a trivial task due to scheduling and dependency constraints on flight tasks.

Sometimes, one task must be interrupted to fly another due to time window constraints. In a scenario with two tasks

Data: Priority queue W, Flight task T, Branch factor B, Max turn angle θ, Waypoint interval δ
Result: Sequence of positions (a sub-flight)
W.insert(node(T.startPosition, T.startOrientation), 0);
while *W not empty* **do**
  node,score = W.pop();
  if *score* $\geq$ *T.desiredScore* **then**
    return Traceback(node);
  end
  for i *in* $[-B, B]$ **do**
    newOrientation = node.orientation + θ * ($\frac{i}{B}$);
    newPosition = node.position + $\delta(\cos(newOrientation), \sin(newOrientation))$;
    newNode = (newPosition, newOrientation);
    newNode.parent = node;
    newScore = T.RewardFunction(newNode);
    W.insert(newNode, newScore);
  end
end

Algorithm 1: Pseudocode for the greedy sub-flight planner.

Task Type	Reward Function
Coverage	The task area is discretized on latitude/longitude aligned grid with spacing controlled by granularity parameter. Candidate sub-flights are given 1.0 unit of reward for each discretized point they come within a threshold of. The first unreached point in the list gives a reward proportional to the sub-flight's proximity using a Gaussian function.
Sampling	The sampling task is parameterized by s, the number of seconds that the sub-flight must spend within the task area. Candidate sub-flights are given 1.0 unit of reward for every second spent within the task area for up to s seconds.
Fly-Through[1]	Returns a fixed reward when candidate sub-flights fly within it at all. Equivalent to a Sampling task configured with a very small value of s.

TABLE II
THE GREEDY SUB-FLIGHT PLANNER USES A DIFFERENT REWARD FUNCTION FOR THE DIFFERENT TASK TYPES.

α and β each requiring 10 seconds to be completed, for example, task α may have the constraint that it must be completed entirely after 5 seconds but before 20 seconds (a window longer than 10 seconds is required due to the time taken to fly between tasks). Task β would be scheduled for at least the first 5 seconds, after which a transition to task α would be made. After task α was completed at around 15-20 seconds, task β would be scheduled again and finished.

We attack the problem by returning to the CPU scheduling metaphor. Flight tasks are processes and the UAV is a CPU. The UAV can fly a task to completion or it can run it in time slices with other tasks, "context-switching" between them. However, flight task scheduling has two key differences from CPU scheduling: First, we know exactly how much time each flight task needs since sub-flights which satisfy them were calculated previously, whereas the run time of a program cannot generally be predicted. Second, the time required to transition or "context switch" between flight tasks is highly variable and must be considered.

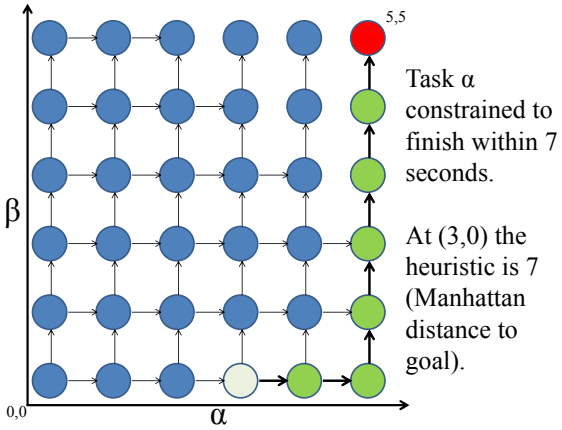

Fig. 5. The scheduling state space of an imaginary two-task planning problem. Both tasks (α and β) take five seconds to be completed. In this example, task α has a time window constraint specifying that it must be finished by no later than seven seconds. Edges that would allow task α to make progress outside of its valid time window have been cut. The A* heuristic is Manhattan distance to the goal. At node (3,0) the value of the heuristic is 7.

We treat the scheduling problem as a graph search through an n-dimensional scheduling space where n is the number of flight tasks in the planning problem. There is previous work in CPU and manufacturing scheduling that also uses graph search [18], [19]. The goal of the search is to find a least-cost path from $(0_1, ..., 0_n)$ to $(c_1, ..., c_n)$ where c_i is the amount of time required to complete sub-flight i. A problem of three tasks each taking 10.5 seconds, for example, requires the scheduler to find a least-cost path from $(0, 0, 0)$ to $(10.5, 10.5, 10.5)$. Note that the total time required to fly the resulting flight plan will be greater than the sum of the sub-flights' required times because flying transitions between tasks takes time.

Solving the scheduling problem can be done optimally using A*. This is a good choice because we can use the distance from any state to the goal as an admissible heuristic. Since the CPU scheduling metaphor restricts our UAV to flying one sub-flight at a time, the scheduler can only do state transitions along one dimension at a time within the scheduling space. This means that all A* distance calculations in the scheduler, including the heuristic, use a "Manhattan" rather than Euclidean distance. See Figure 5.

When the A* scheduler considers transitioning from one flight task α to another task β it must generate a transition (or intermediate) flight from the current state's progress along α's sub-flight to the generated state's progress along β's sub-flight. If the A* scheduler for two tasks α and β had reached node (3,0) and wanted to explore node (3,1), for example, it would use the intermediate planner to generate a flight from the UAV's configuration 3 seconds into α's sub-flight to the beginning (time 0) of β's sub-flight. The length of the intermediate flight is used to calculate how long it will take to transition from α to β and is A*'s "cost to move". Intermediate flights are stored during scheduling so that they can be used as components of the overall flight when scheduling is completed.

Flight task scheduling constraints such as dependencies

and valid time windows are encoded in the state space as obstacles, which A* can easily deal with. Specifically, the search is not allowed to expand into states where any task's dependency constraints or valid time windows are violated. It is important to note that it is very easy to create unsolvable planning problems by creating a dependency loop among two or more tasks or by specifying unrealistic time windows. In this situation the scheduler will search as far as it can before detecting that the problem is overconstrained. At that point, the user can reformulate their planning problem.

D. Intermediate Planner

The intermediate planner is responsible for planning flights from the start position to task areas and in-between task areas. Intermediate flights are planned solely to get the UAV from one configuration to another rather than to accomplish any of the flight tasks. The intermediate planner takes as input the positions of no-fly zones and starting and ending positions/orientations. It outputs a series of positions or waypoints.

The requirements for the intermediate planner are challenging. First, it must be very efficient because it runs as a subroutine of the scheduler's A* search. Second, it must be able to plan over long distances. Third, it must avoid no-fly zones. Finally, it has to be able to plan intermediate flights that are valid according to the UAV's kinematic constraints and that start and end in the correct orientations (not just positions).

Our intermediate planner, like our overall planner, takes a hierarchical approach. As a first step it uses A* to generate a coarse, high-level, obstacle-avoiding path without consideration for desired orientation or UAV kinematics. The A* search is carried out on a coarse 4-connected graph with node spacing of 300 meters so that it will run quickly. Next, it strings the nodes of the A* search together using Dubins Curves [20], which ensure that the desired starting and ending orientations are honored and that UAV kinematic constraints are obeyed. Future work should validate that Dubins curves are an appropriate model for a fixed-wing UAV.

V. Results and Limitations

We tested our planning system on several simulated wilderness search and rescue (WiSAR) scenarios (see for example [21]–[23]). These scenarios tested the ability of the system to plan sub-flights for different tasks, schedule them based on dependency and time window constraints, and build an overall flight while avoiding no-fly zones.

Our first simulated WiSAR scenario is that a lost child was last seen along a popular trail in a canyon. The goal of the flight is to obtain aerial imagery of the trail along the canyon floor for the search team. The secondary goal is to maintain communications with another group of searchers by flying to an area within radio range to exchange messages. Finally, the UAV must return to its launch point and avoid flying near the dangers of the steep canyon walls.

We encoded this scenario in our planning framework using three flight tasks in three separate areas, in addition to a no-fly zone. First, a coverage task gathers aerial imagery of the trail along the canyon floor. Second, a sampling task within communications range serves to relay messages between search groups. Third, a fly-through task near the UAV's starting point brings the UAV back to the launch area. The sampling task is configured with a dependency on the coverage task and the fly-through task is configured with a dependency on both other tasks to ensure that our priorities are encoded in the generated flight. Finally, a no-fly zone is placed over the dangers of the steep canyon wall between the coverage and sampling tasks to ensure that the UAV does not crash.

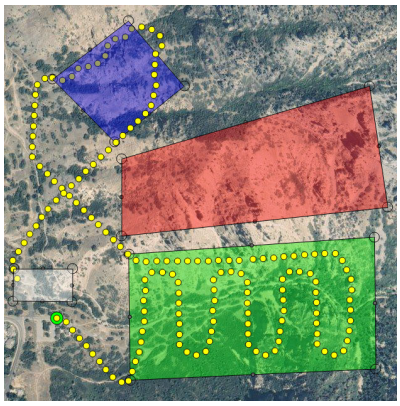

Fig. 6. The results of planning on our first simulated scenario. The green polygon is the coverage task. The red polygon is the no-fly zone. The blue polygon is the sampling task. The white polygon is the fly-through task. The planned flight is represented by the series of yellow dots starting at the large green dot (the starting point).

The results of running the planning algorithm on the first scenario are shown in Figure 6. The planner succeeded in satisfying all three tasks in the correct, desired order while avoiding the no-fly zone.

The second simulated scenario is that of an overdue hiker who did not leave information on which route they would be taking to their destination. Search personnel want to gather aerial imagery of three likely routes as quickly as possible. The search team has conflicting information and believes that all three search areas are equally important to the search.

We encoded this scenario in the planner as three coverage tasks to gather imagery of the search areas and a fly-through task to bring the UAV back to its starting area. Since the search team has assigned equal importance to each area, none of them are selected to be searched before the others. Instead, the planner is left to find the most efficient flight it can. The results of the planner on this scenario are shown in Figure 7. The planner took about 10 seconds to generate this flight on an Intel i7 running at 2.67 GHz.

In summary, our sensor-driven hierarchical planner has shown the ability to generate flights which satisfy complex scheduling constraints while accomplishing sensor tasks. Generated flights are also fairly efficient despite the greedy search method used in the sub-flight planner.

Running times are very good until the number of tasks grows beyond four or five, at which point they increases quickly. The bottleneck in the performance of our planner seems to be the scheduler. The dimensionality of the schedul-

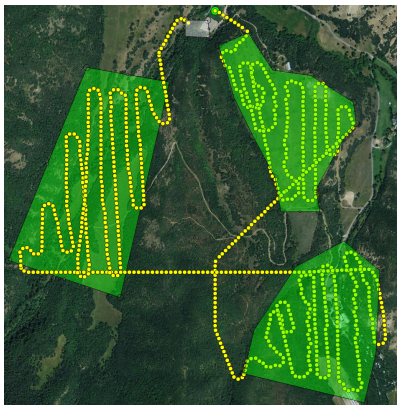

Fig. 7. The results of planning our second simulated scenario. The three green polygons are coverage tasks. The small white rectangle is a fly-through task. The planner was able to generate a flight that satisfied all three coverage tasks in a reasonable order and then returned to the starting area. This flight would take approximately 22 minutes for a small UAV to fly at 14 m/s.

ing space can easily grow to be a problem for A* since each flight task is a dimension of the scheduling space. It is possible that alternative scheduling algorithms may be better able to cope with the high dimensionality. We believe that performance is acceptable for now since most flights do not have a large number of tasks.

VI. Conclusions

We've created a taxonomy of canonical UAV tasks and sensors. Using this taxonomy's generalizing power, we've created a hierarchical algorithm for sensor-driven planning. The algorithm breaks the problem into planning and scheduling components and attacks them separately. It is capable of satisfying complicated scheduling constraints while honoring a simple model of UAV kinematics and no-fly zones. This algorithm was implemented with a prototype user interface and tested on several simulated WiSAR scenarios. During tests, it quickly generated reasonable flights which satisfied all scheduling constraints and no-fly zones.

VII. Future Work

There are a number of interesting directions that future work could take. First and foremost, extending this work to three dimensions would be beneficial, especially with the addition of terrain modeling. It would also be very useful to extend our algorithm to support gimbal planning since many UAV-carried cameras are mounted on movable gimbals. Further extensions could include multi-UAV support, improved UAV kinematics, and improved scheduling.

References

[1] J. Rubio, J. Vagners, and R. Rysdyk, "Adaptive path planning for autonomous uav oceanic search missions," in *AIAA 1st Intelligent Systems Technical Conference*, 2004, pp. 20–22.

[2] W. Zang, J. Lin, Y. Wang, and H. Tao, "Investigating small-scale water pollution with uav remote sensing technology," in *World Automation Congress (WAC), 2012*, June, pp. 1–4.

[3] M. Kontitsis, K. Valavanis, and N. Tsourveloudis, "A uav vision system for airborne surveillance," in *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, vol. 1, April-1 May, pp. 77–83 Vol.1.

[4] Y. Lin, J. Hyyppa, and A. Jaakkola, "Mini-uav-borne lidar for fine-scale mapping," *Geoscience and Remote Sensing Letters, IEEE*, vol. 8, no. 3, pp. 426–430, May.

[5] E. Zaugg, D. Hudson, and D. Long, "The byu sar: A small, student-built sar for uav operation," in *Geoscience and Remote Sensing Symposium, 2006. IGARSS 2006. IEEE International Conference on*, 31 2006-Aug. 4, pp. 411–414.

[6] G. Stephens, S. Miller, A. Benedetti, R. McCoy, R. McCoy Jr, R. Ellingson, J. Vitko Jr, W. Bolton, T. Tooman, F. Valero, *et al.*, "The department of energy's atmospheric radiation measurement (arm) unmanned aerospace vehicle (uav) program," *Bulletin of the American Meteorological Society*, vol. 81, no. 12, pp. 2915–2938, 2000.

[7] V. M. McHugh, C. S. Harden, D. B. Shoff, B. S. Ince, S. E. Harper, G. E. Blethen, R. J. Schafer, P. Arnold, S. Pavitt, M. Thomas, *et al.*, "Using an array of ion mobility spectrometers for ground truth measurements in field tests involving releases of chemical warfare agent surrogates," *International Journal for Ion Mobility Spectrometry*, vol. 6, pp. 49–52, 2003.

[8] P. Niedfeldt, R. Beard, B. Morse, and S. Pledgie, "Integrated sensor guidance using probability of object identification," in *American Control Conference (ACC), 2010*. IEEE, 2010, pp. 788–793.

[9] F. Bourgault, T. Furukawa, and H. Durrant-Whyte, "Optimal search for a lost target in a bayesian world," in *Field and Service Robotics*, ser. Springer Tracts in Advanced Robotics, S. Yuta, H. Asama, E. Prassler, T. Tsubouchi, and S. Thrun, Eds. Springer Berlin Heidelberg, 2006, vol. 24, pp. 209–222.

[10] L. Lin and M. Goodrich, "Uav intelligent path planning for wilderness search and rescue," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, Oct., pp. 709–714.

[11] P. J. Hardin and M. W. Jackson, "An unmanned aerial vehicle for rangeland photography," *Rangeland Ecology & Management*, vol. 58, no. 4, pp. 439–442, 2005.

[12] D. Caltabiano, G. Muscato, A. Orlando, C. Federico, G. Giudice, and S. Guerrieri, "Architecture of a uav for volcanic gas sampling," in *Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on*, vol. 1, Sept., pp. 6 pp.–744.

[13] M. Quigley, B. Barber, S. Griffiths, and M. Goodrich, "Towards real-world searching with fixed-wing mini-uavs," in *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*. IEEE, 2005, pp. 3028–3033.

[14] D. Rathbun and B. Capozzi, "Evolutionary approaches to path planning through uncertain environments," in *Proc. of AIAA UAV Conference*, 2002.

[15] G. Yang and V. Kapila, "Optimal path planning for unmanned air vehicles with kinematic and tactical constraints," in *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, vol. 2. IEEE, 2002, pp. 1301–1306.

[16] S. Bortoff, "Path planning for uavs," in *American Control Conference, 2000. Proceedings of the 2000*, vol. 1, no. 6. IEEE, 2000, pp. 364–368.

[17] M. Kothari, I. Postlethwaite, and D. Gu, "Multi-uav path planning in obstacle rich environments using rapidly-exploring random trees," in *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*. IEEE, 2009, pp. 3069–3074.

[18] M. B. Jones, D. Roşu, and M.-C. Roşu, "Cpu reservations and time constraints: Efficient, predictable scheduling of independent activities," in *ACM SIGOPS Operating Systems Review*, vol. 31, no. 5. ACM, 1997, pp. 198–211.

[19] D. Y. Lee and F. DiCesare, "Scheduling flexible manufacturing systems using petri nets and heuristic search," *Robotics and Automation, IEEE Transactions on*, vol. 10, no. 2, pp. 123–132, Apr.

[20] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, no. 3, pp. pp. 497–516, 1957. [Online]. Available: http://www.jstor.org/stable/2372560

[21] T. Setnicka, *Wilderness Search and Rescue*. Appalachian Mountain Club, 1980.

[22] R. Koester, *Lost Person Behavior: A Search and Rescue Guide on Where to Look – for Land, Air, and Water*. dbS Productions, 2012.

[23] M. A. Goodrich, B. S. Morse, D. Gerhardt, J. L. Cooper, M. Quigley, J. A. Adams, and C. Humphrey, "Supporting wilderness search and rescue using a camera-equipped mini uav," *Journal of Field Robotics*, vol. 25, no. 1-2, pp. 89–110, 2008. [Online]. Available: http://dx.doi.org/10.1002/rob.20226