Minimizing Turns and Altitude Changes in a 3D Sweeping Coverage Problem for UAVs

Yingzi Zhang
Dept. of Mechanical and Mechatronics Engineering
University of Waterloo
Waterloo, ON N2L 3G1
Email: y689zhan@uwaterloo.ca

Stephen L. Smith
Dept. of Electrical and Computer Engineering
University of Waterloo
Waterloo, ON N2L 3G1
Email: stephen.smith@uwaterloo.ca

Stanislav Bochkarev
Dept. of Electrical and Computer Engineering
University of Waterloo
Waterloo, ON N2L 3G1
Email: stas7775@gmail.com

Abstract*—The sweeping coverage problem has been looked at extensively in literature, particularly in scenarios where UAVs must capture photographs of a large 2D area. We attempt to look at this problem where the UAV can move in 3D instead of being conventionally restricted on a plane. It is important for the UAV to be able to move in 3D when the area to be covered does not have a uniformly-defined image-resolution requirement, but instead, has several sub-areas that need to be captured in higher or lower resolutions. In this paper, we attempt to create an algorithm that allows the UAV to autonomously take images of the coverage area at the correct minimum-required resolutions. While there are some existing algorithms that can do this that minimize the total travel distance of the UAV, there does not seem to be one that minimizes the number of turns the UAV conducts, despite the fact that monitoring applications can benefit from minimal UAV turns in terms of image quality and in time. There is also benefit in minimizing the total number of altitude changes the UAV makes as well in order to minimize image distortion. Thus, we attempt to create a 3D path planning algorithm for the UAV to traverse such that we minimize both the number of turns and the number of total altitude changes. This is done using: a clustering algorithm that minimizes the number of altitude changes by grouping regions of similar image-resolution values under one uniform image-resolution, and a CGTSP tour of the clusters such that rotations are minimized. In addition, we provide a focal length controller to take advantage of the zoom capability of newer UAV cameras to further minimize the number of altitude changes, and prove that it is an optimal algorithm.

I. Introduction

The sweeping coverage problem is a problem in which a robot must plan a path over a certain area such that most or all points in this area are successfully covered by the footprint of the robot. There have been many different types of sweeping coverage algorithms devised and solved for specific types of applications, or to optimize specific attributes, covered in seminal surveys [1] [2]. These types of algorithms see many applications in surveillance [3], photogrammetry [4], demining [5], archaeology [6], search and rescue [7], floor cleaning [8], agriculture [9], and many others. However, many of these coverage path planning algorithms are constricted to a 2D plane, and do not take into consideration of a robot moving in 3D. Consideration of the 3D case is useful in applications such as photogrammetry, where some regions of the coverage area need to be photographed in higher quality than others.

Both UAV rotations and altitude changes can create undesirable effects on path quality [10] and image quality [11], which is why the objective is to minimize the two. In regards to UAV rotations, Huang [10] and Bochkarev and Smith [12] both explain that minimizing the number of turns is better for time and image quality. Some UAVs take a considerable amount of time turning compared to simply moving in a straight direction. Others have no camera stabilizers, so distortions of its images and the field-of-view will arise when capturing photos while turning with a slight roll angle. In regards to UAV altitude changes, constantly changing altitude also degrades image quality, as the change of altitude over a short distance will affect the UAV's pitch angle, which will distort the images. In addition, having the UAV change altitude at every time step would be not energy-efficient, and will affect the size of the UAV footprint on the ground as well, thereby necessitating more rotations to ensure full coverage.

Finally, due to the introduction of optimal zoom capabilities in recently developed commercial UAV cameras, a focal length controller algorithm can be devised to take advantage of this new feature in order to further minimize the number of altitude changes for UAVs with such cameras.

A. Contributions

There are four main contributions in this paper. First, the problem of having a 2D area to cover where each discrete point in the area requires the UAV to be at a different altitude appears to not yet be covered in literature. However, this does not mean there are no existing algorithms that can solve such a problem, especially when the objective is to minimize the total travel distance. These algorithms will be discussed in the II section.

Our second contribution is that our algorithms attempt to look at this 3D coverage problem with the objectives of minimizing the number of rotations and the number of altitude changes. This is done by clustering patches of area with similar minimum-required image-resolution values together, so that the UAV can fly at a constant altitude within each cluster. This constant altitude corresponds to the highest required resolution value in the cluster. After that, each cluster is populated with a minimal number of parallel lines that represents the robot's traversal path, since a minimal number of lines corresponds to a minimum number of rotations, as mentioned by Huang [10]. Then, we compute a tour of these lines.

Computing a tour of these lines involves us solving what we call a "clustered generalized travelling salesman problem" (CGTSP). Both GTSP and CTSP are well known and have approximate solutions. However, the CGTSP problem does not seem to appear in known literature. Our third contribution not only introduces this problem, but we also find a reduction of this problem into the familiar GTSP.

The final contribution of this paper is a greedy focal length controller. We provide proof that this controller minimizes the number of altitude changes given a certain path.

maybe image distortion can just be solved by image processing techniques though...************

II. Related Works

Plonski and Volkan [unpublished] created a 3D path planning algorithm which attempts to minimize the total distance travelled, where the UAV is required to visit a set of upside-down cones where each cone's apex represents a point on the ground. The volume of each cone represents the 3D space where a UAV is able to be in order to take an image with enough resolution of the point on the ground at the cone's apex. Their problem can be extended to ours by representing every image-resolution point in the coverage area with a cone. However, as previously mentioned, their solution will only minimize the total distance travelled.

Sadat et al. [13] considered a similar problem in regards to covering an area that does not have uniform minimum-required resolution. Instead of using a lawnmower-type covering algorithm which we are doing, they use a tree pattern to cover the entire area, where the root node symbolizes the position over Q at which the UAV can take a low-resolution image of the entirety of Q, and where child-nodes can take images of a smaller sub-region with higher resolution. This tree pattern for area coverage is also used in Carpin et al. [14]. They use various traversal algorithms, such as breadth-first, depth-first, and a strategy they devised called the "shortcut heuristic". However, as they mentioned, their algorithm causes the UAV to cover the same regions of Q more than once, which lawnmower-type algorithms avoid doing. Finally, from simulations, they discovered that using a lawnmower-type algorithm at a constant altitude corresponding to the highest resolution in Q actually performs better in terms of minimizing travel distance when there are more sub-regions of high resolution.

Galceran and Carreras [15] created a new coverage algorithm that minimizes elevation changes above the seafloor for an AUV, since moving in the vertical axis is more expensive for most AUVs due to the shape of their chassis being designed for horizontal travel across the seafloor rather than vertical movement. They propose the AUV to first cover contours of the terrain of constant elevation before moving to the next parts of the terrain with a different elevation, to reduce the amount of depth-changing motions the AUV conducts while sweeping the area. As such, their algorithm only minimizes the total number of altitude changes.

Schwager et al. [16] utilized altitude by deploying an optimal configuration of multiple non-moving robots to maximize coverage of an area, where the robots can have different altitudes from each other. In this paper, one of the desires is to minimize the number of rotations the aircraft makes. The MSA algorithm devised by Huang [10] attempted to minimize this number. The MSA algorithm is where area Q is decomposed into the optimal set of convex polygons such that the minimum number of turns is achieved. This is done by covering each sub-regional polygon with parallel lines, representing the traversal path of the UAV, where the lines are oriented perpendicular to the minimum height of the polygon. We will be using a variant of this MSA algorithm in the next section. Finally, we also plan to use a GTSP solver, similar to Bochkarev and Smith [12], to obtain the order in which these lines are traversed. Note that this GTSP solver solves the order of traversal in a way where the UAV does not have to necessarily completely cover one polygon before going to the next.

III. Preliminaries

Let Q be a convex polygonal area we are trying to cover. Let q be a coordinate (x, y) inside Q, $q \in Q$. The user can provide a minimum required resolution per area over Q, called $\Psi(q)$.

Let b be the focal length of the lens. As with most cameras, we assume that there will be a maximum threshold focal length and a minimum threshold focal length of the camera, b_{max} and b_{min}. Since most cameras lose image quality at the extremes of their focal length range, these two variables should be set by the UAV camera operator to be values within the actual maximum and minimum focal lengths of the camera.

Let z be the distance from the focal point of the camera to the ground. z can also be defined as the altitude of the UAV above the ground.

We also define $g(b)$ as the resolution per area over q given the current altitude of the UAV and a given b value. The b value is inputted into this function because b is a variable value in the range: $b_{\text{min}} < b < b_{\text{max}}$. Notice that $g(b)$ does not take q as an input argument. This is because the area covered will be a function of the input variable b.

IV. Minimizing the Number of Altitude Changes and the Number of Turns

The initial problem mentioned before is that minimizing the number of altitude changes directly affects the number of turns. If an altitude change made the UAV fly lower near the ground, then the UAV will have a smaller footprint, so it would need more turns to fully cover the same amount of area compared to if the UAV was flying higher.

Thus, we ultimately would like to minimize the cost function:

$$C = (\text{\# of turns}) + \alpha(\text{\# altitude changes}) \tag{1}$$

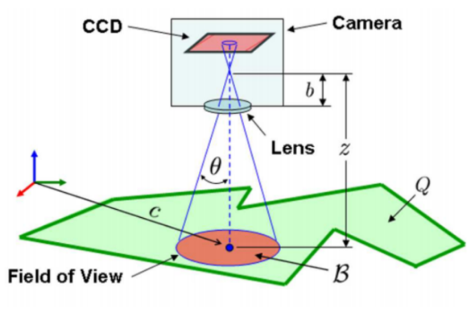

Fig. 1. Diagram of Major Parameters (from MIT Paper)

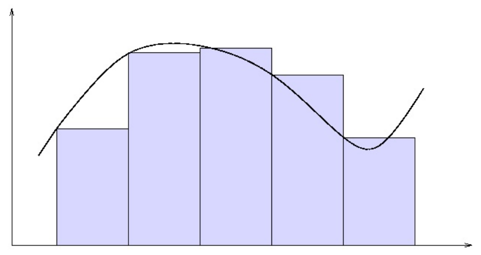

Fig. 2. 2D Visualization of Q Discretized into Sub-Regions of Uniform Altitude (from Google Images). Note that this is not an accurate piecewise decomposition, as some of the piecewise parts are below the maximum $\Psi(q)$ value from the continuous version of the function.

If α is a large number, then minimizing the number of altitude changes becomes a priority, so an ideal method would be to use a clustering algorithm approach to discretize Q into the minimum number of regions of similar $\Psi(q)$ values, such that the UAV needs to only fly at a constant altitude in each of the discretized regions. Afterwards, the path traversal can be determined using the MSA algorithm to cover each discretized region with a minimal number of parallel path lines. Further reasoning behind using a clustering algorithm is detailed in Section V.

If α is a small number, then minimizing the number of turns becomes the priority, rather than the number of altitude changes, which means it is possible to initially use a decomposition method mentioned in previously referenced papers, and then afterwards move some of the edges of the polygons around such that the variance of $\Psi(q)$ is relatively small in each polygon. Examples include the exponential-time polygonal decomposition algorithm, described in the same paper alongside the MSA algorithm by Huang [10], the re-optimization technique by Bochkarev and Smith [12], or Yu's algorithm [17].

We propose using the first method, as we believe the number of altitude changes is of great importance. This algorithm can be seen in Algorithm 1. Essentially, We partition Q into K subregions of uniform altitude using a clustering algorithm, and then conduct Huang's MSA algorithm [10] so that each subregion will be filled with the minimum number of parallel lines. The runtime of this algorithm is $O(c+Km)$, where c is the runtime of the clustering algorithm, and m is the runtime of the MSA function. Section V further describes exactly how this clustering algorithm works.

Algorithm 1 Hybrid MSA and Clustering Algorithm

```
1: $K$ = ClusteringAlgorithm() {Refer to Algorithms 2 or 3}
2: for $k \in K$ do
3:     MSA($k$) {MSA($k$) fills polygon $k$ with parallel lines
       oriented to minimize the number of these lines.}
4: end for
```

V. Minimizing the Number of Altitude Changes by Discretizing $\Psi(q)$ via a Clustering Algorithm

This section attempts to explain the clustering algorithm to decompose area Q in order to minimize the number of altitude changes.

We first define a Lipschitz continuous function. A function is Lipschitz continuous if, for points x_1 and x_2 in the domain of function f, $\forall x_1, x_2$:

$$|f(x_1) - f(x_2)| \leq L|x_1 - x_2|$$

where $L > 0$. The Lipschitz principle can be helpful in terms of defining the variance of $\Psi(q)$ over Q:

$$\begin{aligned} &|\Psi(q_1) - \Psi(q_2)| \leq L|q_1 - q_2|, \\ &\quad \text{where } |q_1 - q_2| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \end{aligned} \quad (2)$$

In other words, there does not exist any sudden huge variances in minimum resolutions from one q to another. From this point on, the assumption is that $\Psi(q)$ is Lipschitz continuous or at the very least, $\Psi(q)$ has natural clusters, where within each cluster, $\Psi(q)$ is Lipschitz continuous. The latter assumption is more reasonable in urban areas. For example, $\Psi(q)$ at the perimeter of buildings will have a sharp jump, but $\Psi(q)$ will not have huge variance within the area that the building takes up.

Using this assumption, we can reduce the number of changes in altitude over the area by collecting areas of similar resolutions under one uniform resolution. For example, if a sub-region has a range of minimum required resolutions between 5 to 10 pixels per square inch, then we can envelop this whole sub-region as a polygonal area with a minimum required resolution of 10 pixels per square inch. An example of this in the 2D case is seen in Figure 2.

Thus, our objective function to minimize is written below, where C_a is the cost function, K is the number of clusters of uniform altitude to partition Q into, and β is the weight for K. P_i is the ith polygon, where $P_1 \cup P_2$ ... $P_k = Q$. Ψ_{μ_i} is defined to be the average $\Psi(q)$ value in P_i. Finally, A_i is defined to be the area of P_i. Note that the second term in the equation attempts to minimize the variance of $\Psi(q)$ within each cluster, while the first term attempts to minimize the number of clusters, which is directly proportional to the number of altitude changes.

$$C_a = \beta K + \sum_{i=1}^{K} \frac{1}{A_i} \int_{q \in P_i} (\Psi(q) - \Psi_{\mu_i})^2 dq \tag{3}$$

There are several problems to overcome in order to minimize this cost function. First, β needs to be found experimentally, similar to finding the learning rate parameter in traditional gradient descent via experiments. Another easier solution would be to have K pre-defined, similar to the k-means algorithm, where the number of clusters, K, is pre-defined, or to pre-define K by solving what K is heuristically, like in the case of Sugar and James [18], where they found K via an information theoretic approach.

Regardless of the method, algorithms in finding the exact polygonal decomposition of Q to minimize Equation 3 are shown in Algorithms 2 and 3. Algorithm 2 shows how to find the clusters when β is known, and Algorithm 3 shows how to find the clusters when K is known. In both methods, $\Psi(q)$ has been transformed into a discrete function in order to be easier to work with. As such, the transformed cost function in Algorithm 2 is:

$$C_{a_{discrete_1}} = \beta K + \sum_{i=1}^{K} \sum_{q \in P_i} \frac{(\Psi(q) - \Psi_{\mu_i})^2}{N_i} \tag{4}$$

Where N_i is the number of $\Psi(q)$ observations in P_i, and the transformed cost function in Algorithm 3 is:

$$C_{a_{discrete_2}} = \sum_{i=1}^{K} \sum_{q \in P_i} \frac{(\Psi(q) - \Psi_{\mu_i})^2}{N_i} \tag{5}$$

Where the βK term is omitted. For each of these two algorithms, there are common variables that are defined as follows. Let D_* be the list of subregions obtained from the clustering algorithm with the minimum C_a. Let O be the number of unique values of $\psi(q) \in \Psi(q)$.

Essentially, both algorithms initially cluster similar $\psi(q)$ values into buckets, and within each bucket, a connected components algorithm is used in order to further cluster the $\psi(q)$ values within the same bucket based on which are connected to each other physically in the x, y-plane of Q. Subsequently, each $\psi(q)$ value is implied to be stored in an unweighted graph data-structure, where each vertex represents each $\psi(q)$ value. However, the issue with these algorithms is finding the correct number of buckets to sort the $\psi(q)$ values into such that the minimum cost function is realized. In both algorithms, all possible number of buckets are tested using the first for-loop. The only difference between the two algorithms is how C is calculated.

The time complexity of both algorithms is determined as follows, where n is the number of $\psi(q)$ values, v is the number of vertices, which is equivalent to n, and e is the number of edges:

1) The first outermost for-loop is $O(n)$.

Algorithm 2 ClusteringAlgorithm() (where β is known)

```
1: $D_* = \emptyset$
2: $C = \infty$ {Where $C$ is a variable that stores the current minimum cost function}
3: for $i = 1...O$ do
4:   $v = 0$ {where $v$ is the sum of squared errors obtained from using $i$ number of buckets}
5:   $D = \emptyset$ {where D is the set of subregions obtained from using $i$ number of buckets}
6:   Sort every $\psi(q) \in \Psi(q)$ into $i$ buckets in Buckets.
7:   for $j = 1...i$ do
8:     P = connectedComponents(Buckets[$j$])
9:     for $p \in$ P do
10:      $D = D + p$
11:      $v = v + \sum_{k=1}^{N} \frac{(\psi(q)-\mu)^2}{N}$ {where $N$ is the number of $\psi(q)$ points in $p$}
12:    end for
13:  end for
14:  if $v + \beta(D.\text{size}) < C$ then
15:    $C = v + \beta(D.\text{size})$
16:    $D_* = D$
17:  end if
18: end for
19: return $D_*$
```

Algorithm 3 ClusteringAlgorithm() (where K is known)

```
1: $D_* = \emptyset$
2: $C = \infty$ {Where $C$ is a variable that stores the current minimum cost function}
3: for $i = 1...O$ do
4:   $v = 0$ {where $v$ is the sum of squared errors obtained from using $i$ number of buckets}
5:   $D = \emptyset$ {where D is the set of subregions obtained from using $i$ number of buckets}
6:   Sort every $\psi(q) \in \Psi(q)$ into $i$ buckets in Buckets.
7:   for $j = 1...i$ do
8:     P = connectedComponents(Buckets[$j$])
9:     for $p \in$ P do
10:      $D = D + p$
11:      $v = v + \sum_{k=1}^{N} \frac{(\psi(q)-\mu)^2}{N}$ {where $N$ is the number of $\psi(q)$ points in $p$}
12:    end for
13:  end for
14:  if $v < C$ and $(D.\text{size}) = K$ then
15:    $C = v$
16:    $D_* = D$
17:  end if
18: end for
19: return $D_*$
```

2) Line 6 is $O(n)$.
3) The second and third inner for-loops together are composed of the following complexities:
   a) The second for-loop by itself is O(n).
   b) From $j = 1$ to i we repeatedly call the connectedComponents function, so after this second for-loop ends, the whole graph will have been traversed. Thus, we can think of the call of all connectedComponents functions to be the equivalent of a programming statement of time complexity $O(v + e)$ outside of this second for-loop. We can extend this further: v is equal to n, and the worst number of e is when every component is connected to every other component in each bucket (from bucket 1 to bucket i), which is $O(4\sum_{i=1}^{\sqrt{n}-1} i) \approx O(\sum_{i=1}^{\sqrt{n}} i) \approx O(\frac{\sqrt{n}(\sqrt{n}+1)}{2}) \approx O(n)$ (assuming a vertex can only be connected to another vertex in one of the four cardinal directions). Thus, $O(v + e) \approx O(n + n) \approx O(n)$.
   c) In the worst case scenario for the third for-loop, every $\psi(q)$ value will be positioned so that none will be connected to its adjacent $\psi(q)$ values, even when there are as little as two buckets (i.e. when the $\psi(q)$ values form a pattern in which they alternate between buckets 1 and 2). Thus, the third inner for-loop is $O(n)$.
   d) We can treat the summation on line 11 as either a constant, or we can sum up all the times we conduct this summation, which we can think of as an O(n) complexity outside of the second for-loop (this is because we do the variance calculation for all n number of points).

From the list of reasons above, the expanded complexity of both algorithms is: $O(n[n + n + n(n) + n])$. From left to right: the first n represents the outermost for-loop, the second is from line 6, the third is from $O(v+e)$, the fourth and fifth represent the second and third for-loops, and the final one represents the summation of variance calculations for all n points. Simplifying this complexity expression gives the final complexity of $O(n^3)$. Despite the time complexity being this large, these two algorithms are guaranteed to find the global minimum, where in one case, the global minimum is found subject to β being a known constant, and in the other case, the global minimum is found subject to K being a known constant.

A. Hierarchical Clustering via Ward's Method

Another possible way to partition Q into disjoint clusters, instead of obtaining a user-defined K or β, is to obtain a user-defined threshold value for the maximum intra-cluster variance, J. By knowing this, we can define a similar, but new problem in minimizing the number of altitude changes compared to the previous two algorithms:

Given $G = (V, E)$ as the unweighted connected graph that represents each $\Psi(q)$ point in Q, where every $\psi(q) \in V$, and E is the set of edges that represent the adjacency of the $\psi(q)$ values in Q, minimize K such that $V_1, V_2, ..., V_K$ are connected subsets of V, $V_1 \cup V_2 \cup ... \cup V_K = V$, and $variance(V_i) \leq J$ for $i = 1...K$.

We believe that allowing the user to choose J instead of K and β is better in a practical perspective, as image quality and field-of-view is greatly dependent on intra-cluster variance, and it will be difficult for the user to discern what appropriate values K and especially β should be for their use.

This problem can be solved heuristically by utilizing a form of hierarchical clustering. Hierarchical clustering can be either agglomerative or divisive. Agglomerative hierarchical clustering is a bottom-up approach, where individual $\Psi(q)$ values are initialized as clusters of their own, and with each iteration of the algorithm, smaller clusters form into larger clusters, until a stopping point is reached. Divisive hierarchical clustering is a top-down approach, where the area Q is divided into two or more large regions, and those regions are subsequently divided further. We decided to use a bottom-up approach, as it ensures from the start to the finish that only points that are very similar in $\Psi(q)$ value to be merged together. In contrast, in a top-down approach, once we make a globular cut to divide a large cluster into two, it cannot be restitched again, which causes some concerns in terms of minimizing variance in each cluster, especially once the clusters gradually become smaller. We can use Ward's method for agglomerative clustering [19], where in each iteration of the algorithm, only the two clusters with the minimum increase in sum-of-squared errors will be merged. To put it more formally, a merge of two clusters C_i and C_j only occurs when the pair, C_i and C_j, specifically results in the lowest value of the following objective function, out of all cluster-pair combinations:

$$\begin{aligned}\Delta(C_i, C_j) = &\sum_{k \in C_i \cup C_j} (\psi_k - \mu_{C_i \cup C_j})^2 \\ &- \Big(\sum_{k \in C_i} (\psi_k - \mu_{C_i})^2 + \sum_{k \in C_j} (\psi_k - \mu_{C_j})^2\Big) \quad (6)\end{aligned}$$

$$\Delta(C_i, C_j) = \frac{n_{C_i} n_{C_j}}{n_{C_i} + n_{C_j}} (\mu_{C_i} - \mu_{C_j})^2 \quad (7)$$

Where μ_{C_i} corresponds to the average Ψ value in cluster C_i.

Agglomerative clustering algorithms, including Ward's algorithm, needs a defined stopping point or else the algorithm will continue until everything is all in one cluster. For our application, we can determine this stopping point by using the user-defined J value. As soon as a merged cluster crosses the J variance threshold, we stop the algorithm.

The clustering algorithm via Ward's method for our specific application is shown in Algorithm 4. Note that M is a symmetric matrix that stores and keeps track of $\Delta(C_i, C_j)$ for every cluster-pair combination. Also note that if two clusters are not adjacent to each other, then $\Delta(C_i, C_j) = \infty$. The time complexity for Algorithm 4 is $O(n^2)$.

B. Clustering Algorithm Lower Bound

It is evident that we cannot find a guarantee to this clustering algorithm without making heavy assumptions on the connectivity of the Ψ graph in Q. Therefore, we attempt to output a

Algorithm 4 ClusteringAlgorithm() (where the intra-cluster variance, J, is known)

```
1: Initialize each $\psi(q) \in \Psi(q)$ as its own cluster $c \in C$.
2: Initialize graph $G$. {G holds every $\Psi(q)$ value and its adjacent relationships to other $\Psi(q)$ values.}
3: Initialize $J$ to be a user-defined maximum variance value.
4: Initialize $currMaxVar \leftarrow 0$.
5: while $C.size > 1$ do
6:     Initialize $minDelta \leftarrow \infty$.
7:     Initialize $cluster_1 \leftarrow \emptyset$ and $cluster_2 \leftarrow \emptyset$.
8:     for $i = 1$ to $n$ do {finding the smallest $\Delta$ between two clusters.}
9:         $M \leftarrow$ minDeltaWith($i$) {where minDeltaWith($i$) returns the minimum of the four distances between the four adjacent $\Psi(q)$ values with the $i$th $\Psi(q)$ value}
10:        if $M \leq minDelta$ then
11:            $minDelta \leftarrow M(i,j)$
12:            $cluster_1 \leftarrow i$
13:            $cluster_2 \leftarrow j$
14:        end if
15:    end for
16:    if variance($cluster_1$ and $cluster_2$) $\geq J$ then
17:        return {Done merging clusters.}
18:    end if
19:    mergeClusters($cluster_1$, $cluster_2$)
20:    Update $G$ such that $cluster_1$ and $cluster_2$ are one node.
21: end while
```

lower bound for K by relaxing the connectivity constraint, and only cluster similar Ψ values together, regardless of whether or not two similar Ψ values are necessarily adjacent in Q. To do this, J and all ψ points in Q, Ψ, must be given. This lower bound essentially is the specific scenario when all Ψ values that are very similar to each other are all coincidentally adjacent to each other. There is no guarantee that our algorithm will give a value of K close to this lower bound. However, outputting this lower bound and comparing it with the outputted K from our algorithm can be of use, and with experimental evidence, we can see if our clustering algorithm can be within a small constant factor from the lower bound.

This lower bound on K can be found by Algorithm 5, where J and a sorted list Ψ are given. Essentially, this algorithm goes through the sorted list of Ψ from beginning to end and greedily clusters the Ψ values. Note that the Ψ-range of every cluster is the same and must be equal to J.

1) Proof of Correctness for Lower Bound: The logic that we use is very similar to the proof of correctness we use for our greedy focal length controller.

To prove the optimality of our algorithm, we must first prove that our algorithm exhibits both the greedy choice property and the optimal substructure property.

Lemma V.1. *Let us define the first step of our algorithm, d_1. d_1 finds the first interval, $[\psi_1, \psi_i]$, that represents the first cluster we greedily find based off of the first Ψ value in the sorted Ψ list, ψ_1. Then, there always exists an optimal solution S^* in which d_1 is its first step.*

Algorithm 5 LowerBound(J, Ψ)

```
1: $tempPsi \leftarrow \Psi[1]$
2: $lowerBoundK \leftarrow 1$
3: for $\psi \in \Psi$ do
4:     if $\psi > tempPsi + J$ then
5:         $tempPsi \leftarrow \psi$
6:         $lowerBoundK \leftarrow lowerBoundK + 1$
7:     end if
8: end for
9: return $lowerBoundK$
```

The following proves this greedy choice property. Let R be defined as one optimal solution. If R already includes d_1, then the property holds and is trivial. If R does not include d_1, then there is only one possibility: the first step in R starts at a value of Ψ less than ψ_1. The first cluster cannot start at a Ψ value greater than ψ_1 or else ψ_1 will not be in any cluster at all. Let us call this first step of R as d_{R_1}.

We can exchange d_{R_1} that was originally from R with our first step, d_1, very easily, even if the second cluster in R, d_{R_2}, is directly adjacent to d_{R_1}. We can simply remove d_{R_1}, shorten d_{R_2} such that its interval starts right after ψ_i, where ψ_i is the value d_1 ends, and then finally place d_1 into R. Thus, the following holds true: d_{R_1} can be exchanged with d_1 in R, to produce another solution, S^*, which is still optimal.

Lemma V.2. *Let S' denote the sequence of a minimum number of disjoint clusters, where the total range of all the clusters is between $(\psi_i, \psi_n]$, where ψ_i is the last point in the first cluster (whatever the first cluster may be), and where ψ_n is the last Ψ value in the sorted list. Then, $S = \langle d_1, S' \rangle$ must be an optimal solution.*

We now prove this optimal substructure property. We know that none of the clusters in sequence S' overlap with d_1. Thus, $S = \langle d_1, S' \rangle$ is a feasible solution. We also know from the last lemma, there exists an optimal sequence S^* in which d_1 is its first cluster. It is apparent that $S^* - d_1$ is also compatible with d_1. Since S' was defined to be an optimal sequence for $[q_i, q_n]$, then the following inequality holds:

$$|S'| \leq |S^* - d_1|$$

Thus, we can apply the following logic:

$$\begin{gathered} |S^* - d_1| = |S^*| - 1 \\ |S'| \leq |S^*| - 1 \\ |S| = |S'| + 1 \\ |S| - 1 = |S'| \\ |S| - 1 \leq |S^*| - 1 \\ |S| \leq |S^*| \end{gathered}$$

Thus, S is an optimal solution.

Finally, we can prove the correctness of this algorithm by induction:

1) Basis: If $|S| = 1$, the algorithm gives us an optimal solution, as d_1 is already optimal.
2) Induction Step: Assume our algorithm can give us optimal solutions for all sequences where the number of clusters is z, for some $z \geq 1$. We show that our algorithm also provides an optimal solution for sequences of clusters of size $z + 1$. Consider a coverage path where the optimal solution results in $z + 1$ intervals. By the greedy choice property, d_1, obtained from our algorithm, is part of some optimal solution for this problem. By the optimal substructure property, after d_1, there is an S' that can be combined with d_1 to give us an optimal solution, $S = \langle d_1, S' \rangle$. S' has to consist of z steps. We already assumed we can solve problems of size z, and thus, using the logic above, we can also solve S, or more generally, problems of size $z + 1$.

C. User-Defined Polygonal Decomposition of $\Psi(q)$

Instead of the above two methods, we could disregard the concept $\Psi(q)$ entirely and instead use a completely different way to describe the minimum required resolution over sub-regions in area Q. We could instead have the end-user manually tell us convex polygons over Q in which there is a uniform minimum resolution required. For example, using a Google Maps-like application, the user can draw polygons over sub-regions of Q where there needs to be better or worse resolution than the rest of Q. If the user requests a concave polygon shape to have a certain $\Psi(q)$ value, we can break the concave polygon into multiple convex polygons in order to get a smaller amount of lines, and thus a smaller number of rotations, than if we were to keep the concave polygon intact—similar to Bochkarev and Smith [12].

VI. Clustered GTSP (CGTSP) Tour Generation

In Bochkarev and Smith [12], a GTSP (generalized travelling salesman problem) solver was used in order to plan a coverage path of disjoint polygons, where within each polygon, straight parallel lines oriented along the polygon's longest height are traversed. For graph $G = (V, E)$ where $V = \{v_i...v_n\}$ is the set of all vertices and $E = \{(v_i, v_j); v_i, v_j \in V, i \neq j\}$ is the set of all edges, the GTSP is defined such that the set V is further subdivided into subsets $V_1...V_k$, where k is the number of subsets, $V_1 \cup V_2 \cup ...V_k = V$, and that for all subsets V_i and V_j, where $i \neq j$, $V_i \cap V_j = \emptyset$. Solving the GTSP requires finding the shortest path such that we visit only one node for every subset. For both Bochkarev and Smith's problem and our problem, each straight line can be traversed in two possible ways. Thus, every vertex should represent one way of traversal for one line, and every subset has a size of 2, and represents the two possible traversal paths of a line.

In contrast to the GTSP, a clustered TSP (CTSP) solver is a bit different. In a CTSP, every time the traveler goes into a cluster, they must travel to every node within the same cluster before moving onto another cluster.

A CGTSP is a combination of the two above problems, and must be solved in our specific problem. To clarify, a CGTSP is where the subsets from a GTSP are clustered such that the travelling salesman needs to travel to every subset within the same cluster of subsets before going onto the next cluster of subsets.

Therefore, our graph $G = (V, E)$ is defined such that each $v_i \in V$ represents a line that the UAV traverses in one of two possible directions. Similar to Bochkarev and Smith's GTSP, each subset in our problem consists of only two nodes, representing the two possible traversal directions. The weight of each edge in E represents the cost of travel for the UAV from one line to another. In our simulations, we utilize Dubins' car model to calculate these edge weights. Every cluster of subsets in this CGTSP represents a polygon obtained from the clustering algorithm mentioned in Section V. These clusters of polygons of similar altitudes need to be consecutively travelled one at a time in order to minimize the number of altitude changes.

A. Reduction of CGTSP into GTSP

No solver for CGTSP exists in literature as of yet. However, the previous two types of problems, GTSP and CTSP, have been solved by reducing the problems to a simple ATSP [20] and TSP problem [21], respectively. The CGTSP solver can be solved in a similar way to these two, by reducing it into a GTSP. The transformation, similar to that of Helsgaun's [21], is described below where V' is the transformed vertex set, V'_i is the ith transformed subset of vertices, and c'_{ij} represents the new transformed cost between vertices v_i and v_j:

(a) $V = V'$
(b) $\{V_1...V_k\} = \{V'_1...V'_k\}$
(c) Define $c'_{ij} = c_{ij}$, when v_i and v_j belong to the same cluster.
(d) Define $c'_{ij} = c_{ij} + M$, when v_i and v_j belong to two different clusters. M is a large constant, where $\sum_{(i,j)\in V} c_{ij} < M < \infty$ [22].

When entering a cluster at a vertex v_i, an optimal GTSP tour will always visit all other vertex sets of the cluster before moving to the next cluster due to the added M term for inter-cluster edges. If we define m as the number of inter-cluster edges the GTSP tour will have, then cost of the tour for the CGTSP is the cost of the GTSP tour subtract mM.

VII. Focal Length Controller to Minimize the Number of Altitude Changes

To figure out when we can simply change the focal length versus changing altitude to get the required resolution over a given area, we have made several assumptions. First, the camera uses parfocal lens rather than varifocal lens. Varifocal lens are used by most standard cameras, where they take a set amount of time to be refocused. Parfocal lens, on the other hand, stays in focus even when the focal length is changed. Thus, parfocal lens is more practical in aerial surveillance for a fixed-wing UAV since there will not be any waiting duration for the camera to refocus while the aircraft is flying over the coverage area. The second assumption we made is that the UAV can accurately estimate its altitude over Q. There are various papers that cover altitude estimation for UAVs, including [23] and [24]. Thirdly, we do not consider digital zoom at all. This section entirely deals with optical zoom. Digital zoom is undesirable as it is analogous to merely cropping the original image and enlarging it, which severely affects resolution. From Figure 1, we can say the following relationship for B:

$$B = \left\{ q \mid \frac{|q-c|}{z} \leq tan\theta \right\}$$

Where q in this case is the q at which the UAV and its lens is centred upon. Both Figure 1 and the above equation are taken from Schwager et al. [16]. We need to first figure out a relationship for the number of pixels per square area of the terrain. Schwager et al. determined the area per pixel relationship instead for their application:

$$f(p,q) = \begin{cases} a(b-z)^2 & \text{for } q \in B \\ \infty & otherwise \end{cases}$$

They obtained this equation by squaring the magnification factor equation from classical optics:

$$M = \frac{b}{b-z} \tag{8}$$

They squared this equation since they needed the magnification factor for not a one-dimensional length, but rather an area. Note that the magnification factor should technically be the ratio between the distance from the CCD (the image) to the lens, s and the distance from the ground to the lens, $b - z$. They employ a slightly different equation than this, because when the object to be captured is extremely far away—also known as "focused to infinity"—we can say that:

$$s \approx b \tag{9}$$

Unfortunately, we cannot use their equation for area per pixel and merely inverse it to get our desired pixel per area relationship. They have included an a value that corresponds to the area per each pixel. We want to find an exact equation for a, or use another approach without utilizing a to find the pixels per a subset area q. We will now attempt to find the number of pixels per area using all of the variables from Section III, given that n is the total number of pixels for the camera:

$$g(b) = \frac{n}{b} \tag{10}$$

Let us give a different representation of B from Schwager et al [16]. We instead define it as the following, assuming the area is of a circular shape:

$$B = \pi(ztan\theta)^2 \tag{11}$$

We can obtain θ by the following: from Figure 3, to project a sharp image of distant objects, S_2 needs to be equal to the focal length F (also defined as b from Figure 1), which is

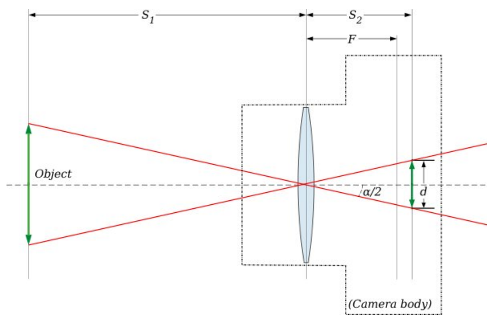

Fig. 3. Relationship between Focal Length, Lens-to-Object Distance, and Lens-to-Image Distance (From Wikipedia)

attained by setting the lens for infinity focus. Then the angle of view, or θ, is given by:

$$\theta = \frac{a}{2} = arctan\left(\frac{d}{2b}\right) \tag{12}$$

Where the constant d is the size of the CCD. Therefore,

$$g(b,z) = \frac{n}{\pi\left(z\left(\frac{d}{2b}\right)\right)^2} \tag{13}$$

In order for a camera to zoom and change the field of view of its image, many try to manually change b (the focal length) via changing the position of the lens inside the camera. This is why we previously defined b_{min} and b_{max}. Note that b_{max} corresponds to the smallest field of view, B, and also the highest resolution per area for a given z, whereas b_{min} corresponds to the biggest field of view and the lowest resolution per area for the same z.

A. *Online Greedy Focal Length Controller*

By finding equation 13, we can now compare the current resolution per area $g(b)$ with the desired minimum resolution per area $\Psi(q)$ to see if we can change focal length only (by plugging in b_{new} into g), or if we need to also change altitude. The online greedy algorithm to figure out whether or not we need to change altitude or simply change focal length is described in Algorithm 6.

The following calculations gives us a relationship between the current focal length and the new desired focal length we need in order for the minimum required resolution to be satisfied, given a constant altitude:

$$g(b_i) = \frac{n}{\pi\left(z\left(\frac{d}{2b}\right)\right)^2} \tag{14}$$

$$g(b_i) = \frac{n}{\pi}\left(\frac{2b_i}{zd}\right)^2 \tag{15}$$

$$g(b_i) = Ab_i^2, \text{ given that } A = \frac{n}{\pi}\left(\frac{2}{zd}\right)^2 \tag{16}$$

$$g_{required}(b_{new}) = \Psi(q+1) = A(b_{new})^2 \tag{17}$$

Algorithm 6 Greedy Online Focal Length Controller

```
1: Let $b$ be the current focal length, and $z$ be the current altitude.
2: for q in Q do
3:     Calculate $g(b, z_{q+1})$ {where $z_{q+1}$ is the altitude over $q+1$}
4:     if $|g(b, z_{q+1}) - \Psi(q+1)| < \epsilon$ then {where $\epsilon$ is the tolerance threshold}
5:         Do nothing. Fly over $q+1$.
6:     else
7:         Calculate $b_{new}$, using equation 20.
8:         if $b_{new} < b_{min}$ or $b_{new} > b_{max}$ then
9:             $b \leftarrow \frac{b_{min}+b_{max}}{2}$
10:            $z \leftarrow \frac{2b}{d}\sqrt{\frac{n}{g(b)\pi}}$
11:        else
12:            $b \leftarrow b_{new}$
13:        end if
14:    end if
15: end for
```

$$\frac{g_{required}(b_{new})}{g(b_i)} = \frac{A(b_{new})^2}{Ab_i^2} \tag{18}$$

$$b_{new} = b_i\sqrt{\frac{g_{required}}{g(b_i)}} \tag{19}$$

$$b_{new} = b_i\sqrt{\frac{\Psi(q+1)}{g(b_i)}} \tag{20}$$

In both cases for automatic polygonal decomposition of continuous $\Psi(q)$ and user-defined polygonal decomposition, this online algorithm for altitude changing can still apply. However, there are several questions, problems, and implications that arise from this initial algorithm.

Firstly, this is an online and greedy method of changing altitude over time. This algorithm does not try to optimize the least of amount of times the altitude changes over all of Q. For example, it may not be optimal to always change the altitude if the threshold is crossed. Perhaps the global optimum of both minimizing number of turns in the path and minimizing the number of altitude changes does not require the altitude to keep changing over an area. Rather, over a sub-region of the area Q, the altitude can remain constant if the minimum resolution required is around the same over this sub-region. We attempt to figure out how to globally minimize the number of altitude changes over Q in the next subsection.

Another problem that remains is what the initial starting altitude should be. We cannot merely start flying at an altitude such that the maximum $\Psi(q)$ value is always fulfilled without changing the altitude afterwards. This leads the risk of leaving the footprint image size to be so small that the UAV would need to turn many times to try to cover the entire area of Q. We want to try to maximize the footprint size so that the minimum number of lines, or the minimum number of turns, is done when covering area Q. A good online initial altitude can thus be the altitude where the minimum required resolution is satisfied when:

$$b_{init} = \frac{b_{max} + b_{min}}{2} \tag{21}$$

b_{init} is an optimal initial focal length in regards to the online, greedy algorithm stated earlier in this section as it gives the UAV maximum local chance to be able to change focal length rather than changing altitude in both lower and upper resolution requirements.

B. Greedy Focal Length Controller

An offline greedy focal length controller was devised instead using the following approach of minimizing a cost function such that we minimize the number of altitude changes.

Let us define a function $f(x)$ where $x \in [0, x_n]$. Let us also define a set $f_1, f_2, ...f_n$ where:

$$f_1(x) = f(x) : x \in [0, x_1] \tag{22}$$
$$f_2(x) = f(x) : x \in (x_1, x_2] \tag{23}$$
$$... \tag{24}$$
$$f_n(x) = f(x) : x \in (x_{n-1}, x_n] \tag{25}$$

so that:

$$f(x) = \begin{cases} f_1(x) & \text{for } [0, x_1] \\ f_2(x) & \text{for } (x_1, x_2] \\ ... & \\ f_n(x) & \text{for } (x_{n-1}, x_n] \end{cases}$$

Also define $g_i = max(f_i(x))$ and $h_i = min(f_i(x))$. Our objective function is to minimize n:

$$C_{focal} = n \qquad \text{such that } \; g_i - h_i \le F \text{ for } \; i = 1...n$$

Where C_{focal} is the cost function, and F is a constant, and wherever the condition $g_i - h_i \le F$ breaks is where f_{i+1} starts. In our specific application, $f(x)$ is $\Psi(q)$, and F is the maximum resolution range that can be achieved just from changing the focal length of the camera. Derived from Equation 16, the maximum resolution range of the camera can be calculated as:

$$F = A(b_{max}^2 - b_{min}^2) \tag{26}$$

Therefore, we attempt to maximize the distance travelled over the coverage path without changing the altitude of the UAV through utilizing the focal range of the camera.

Algorithm 7 demonstrates an offline greedy algorithm to find the optimal initial and all subsequent altitudes to change to throughout the coverage path in order to minimize the cost function. Note that this algorithm assumes we already have a known traversal path to cover area Q from the CGTSP, and that this algorithm is run before the UAV actually travels along the coverage path. The values in the tuple-list object, P, are used to figure out at what points along the coverage path from q_i to q_m, called CP, where the altitude must change, and also the exact value the altitude of the UAV must be changed to. Algorithm 8 uses the tuple list obtained from Algorithm 7 in order to change the altitude of the UAV as the UAV travels along the coverage path in order to minimize the number of altitude changes. Note that the basic principle behind this offline approach is: determine the maximum resolution range we obtain if we purely changed the focal length of the camera, as calculated in Equation 26, and then to look into the future over path CP to see if $(g_i - h_i)$ for a given interval $(q_{i-1}, q_i]$ is larger than the camera resolution range. If there are any q values where this occurs, then we store these q values and calculate what the focal length and the altitude of the UAV should be for all points prior to each of these q values in order to maximize the distance the UAV travels before needing to change altitude.

Finally, the time complexity of both algorithms 7 and 8 is $O(n)$, where n is the number of discrete steps in the coverage path CP. This algorithm not only minimizes the number of altitude changes.

Algorithm 7 Offline Greedy Function to Determine When Altitude Must be Changed

```
1: $P \leftarrow \emptyset$ {where $P$ is a list of tuples. Each tuple has one position value of $q \in Q$, and one altitude value $z$.}
2: $\Psi_{max} \leftarrow \Psi(q_1)$ {where $q_1$ is the first area the UAV will be while traversing along the coverage path.}
3: $\Psi_{min} \leftarrow \Psi(q_1)$
4: for $i = 2$ to $m$ do {where m is the number of ordered elements in $Q$ that make up the traversal path of the UAV, $CP$}
5:     if $\Psi(q_i) > \Psi_{max}$ then
6:         $\Psi_{max} \leftarrow \Psi(q_i)$
7:     end if
8:     if $\Psi(q_i) < \Psi_{min}$ then
9:         $\Psi_{min} \leftarrow \Psi(q_i)$
10:    end if
11:    if $\Psi_{max} - \Psi_{min} \geq A(b_{max}^2 - b_{min}^2)$ then
12:        $b_{mid} = \frac{b_{max}+b_{min}}{2}$
13:        $\Psi_* = \frac{A(b_{max}^2+b_{min}^2)}{2}$
14:        $z = \frac{2b_{mid}}{d}\sqrt{\frac{n}{\Psi_*\pi}}$ {where $z$ in this case is just an altitude value that will be stored in our tuple object $p \in P$}
15:        $p \leftarrow (q_i, z)$
16:        Insert $p$ into $P[P.size]$
17:        $\Psi_{max} \leftarrow \Psi(q_i)$
18:        $\Psi_{min} \leftarrow \Psi(q_i)$
19:    end if
20: end for
```

1) Proof of Correctness: This section establishes the correctness of Algorithms 7 and 8 and proves why this greedy controller produces an optimal number of altitude changes for a given coverage path.

To prove the optimality of our algorithm, we must first prove that our algorithm exhibits both the greedy choice property and the optimal substructure property.

Lemma VII.1. *Let us define the first step of our algorithm, d_1. d_1 finds the first interval, $[q_0, q_i]$, that greedily maximizes the distance travelled until the UAV must change altitude. Then, there always exists an optimal solution S^* in which d_1 is its first step.*

Algorithm 8 Greedy Focal Length Controller

```
1: Let $b$ be the current focal length, and $z$ be the current altitude. Let $P$ be the list of tuples obtained from Algorithm 7.
2: Initialize $b \leftarrow \frac{b_{max}+b_{min}}{2}$
3: for $i = 1$ to $m$ do
4:     if $q_i = P[0].q$ then
5:         $z \leftarrow P[0].z$
6:         Pop $P[0]$ from $P$.
7:     end if
8:     Calculate $b_{new}$, using equation 20.
9:     $b \leftarrow b_{new}$
10: end for
```

The following proves this greedy choice property. Let R be defined as one optimal solution. If R already includes d_1, then the property holds and is trivial. If R does not include d_1, and instead has its first step as d_{R_1} with an interval $[q_{R_0}, q_{R_i}]$, where $d_{R_1} \neq d$, then $q_{R_i} < q_i$, since d_1 was already defined as the maximum interval possible before the maximum camera resolution range condition breaks. From this, we can exchange the original interval $[q_{R_0}, q_{R_i}]$ in R with $[q_0, q_i]$. Note that in order to do so, the interval of difference between d_1 and d_{R_1}, $[q_{R_i}, q_i]$, that was originally in the second step interval in R, d_{R_2}, will move into the first step interval. This transferal process is valid as it does not violate the maximum camera resolution range condition for the second step interval, d_{R_2}. Thus, the following holds true: d_{R_1} can be exchanged with d_1 in R, to produce another solution, S^*, which is still optimal. The only difference now between R and S^* is that the altitude value at which the UAV must fly in the first interval will change, though this does not affect the proof in any way.

Lemma VII.2. *Let S' denote an optimal sequence of intervals between $(q_i, q_n]$, where q_i is the last point in the first interval (whatever the first interval may be), and q_n is the last point in the coverage path. Then, $S = \langle d_1, S' \rangle$ must be an optimal solution.*

We now prove this optimal substructure property. We know that none of the intervals in sequence S' overlap with d_1. Thus, $S = \langle d_1, S' \rangle$ is a feasible solution. We also know from the last lemma, there exists an optimal sequence S^* in which d_1 is its first interval step. It is apparent that $S^* - d_1$ is also compatible with d_1. Since S' was defined to be an optimal sequence for $[q_i, q_n]$, then the following inequality holds:

$$|S'| \leq |S^* - d_1|$$

Thus, we can apply the following logic:

$$|S^* - d_1| = |S^*| - 1$$
$$|S'| \leq |S^*| - 1$$
$$|S| = |S'| + 1$$
$$|S| - 1 = |S'|$$
$$|S| - 1 \leq |S^*| - 1$$
$$|S| \leq |S^*|$$

Thus, S is an optimal solution.

Finally, we can prove the correctness of this algorithm by induction:

1) Basis: If $|S| = 1$, the algorithm gives us an optimal solution, as d_1 is already optimal.
2) Induction Step: Assume our algorithm can give us optimal solutions for all sequences where the number of intervals is z, for some $z \geq 1$. We show that our algorithm also provides an optimal solution for sequences of size $z + 1$. Consider a coverage path where the optimal solution results in $z + 1$ intervals. By the greedy choice property, d_1 is part of some optimal solution for this problem. By the optimal substructure property, after d_1, there is an S' that can be combined with d_1 to give us an optimal solution, $S = \langle d_1, S' \rangle$. S' has to consist of z steps. We already assumed we can solve problems of size z, and thus, using the logic above, we can also solve S, or more generally, problems of size $z + 1$.

2) Minimizing Total Distance in Change of Altitude: Not only does this algorithm minimize the number of altitude changes, but it also minimizes the total distance of change in altitude.

Claim VII.3. *The greedy focal length controller algorithm minimizes the total distance in change of altitude for the UAV.*

We prove this via proof of contradiction. We know that our algorithm produces the minimum number of intervals for path CP from the proof of correctness in the previous section, which directly corresponds to a minimum number of altitude shifts. Let this minimum number of altitude shifts be w. Thus, the only case we have to consider is the case in which there are more than w altitude shifts.

Take some arbitrary interval from point q_i to some point q_j, where $q_j > q_i$, such that our algorithm will have one altitude shift. Define this altitude shift's distance as H. Let us assume there exists another algorithm that gives this interval $[q_i, q_j]$ a larger number of altitude shifts, n, where $n \geq 2$, and where the total distance in altitude change is defined as $H_* = h_1 + h_2 + ... + h_n$. Additionally assume that $H_* < H$, which means $h_1 + h_2 + ... + h_n < H$. However, if this inequality holds true, then there must exist at least one $\Psi(q)$ point along the coverage path that cannot be reached purely changing the focal length of the camera, since we have shortened the total altitude distance to H_*. To compensate for this, we must add more altitude shifts. However, by then, $h_1 + h_2 + ... + h_n + h_{n+1} + ... \geq H$. Thus, by proof of contradiction, there is no such total altitude distance H_*, where $H_* < H$, that can fully cover the desired minimum-required resolutions without changing altitude at another q point between $[q_i, q_j]$. This reasoning can be extended to the entire coverage path, CP, where the following inequality must be true: $H(w+n) \geq H$, where $H(w+n)$ is the total altitude distance when there are $w + n$ number of altitude shifts, and where $n \geq 1, n \in \mathbb{Z}$. Therefore, our algorithm does indeed minimize the total distance in change of altitude.

To summarize, once you decrease the value of H from our algorithm to a smaller value, there will exist $\Psi(q)$ points that cannot be reached without creating additional altitude shifts to compensate. The total H_*, including the additional altitude shifts, will be at least equal to our original H. Thus, our H must be the minimum.

C. Compensating for Other Image Quality Features

Changing focal length indirectly changes other aspects of image quality, which need to be accounted for, such as the f-stop. The f-stop is the ratio of a camera's focal length to the diameter of the entrance pupil:

$$f = \frac{b}{D} \tag{27}$$

Where f is the f-number, b is the focal length, and D is the diameter of the entrance pupil. The f-stop number conveniently summarizes the shutter speed of the camera, and the brightness of the image taken. The brightness of the image is proportional to the inverse of the square of the f-number:

$$Brightness \propto \frac{1}{f^2} \tag{28}$$

Thus, to compensate for the darkened image, the camera will need a longer exposure time to collect more light, which decreases the shutter speed. However, in our application, a slower shutter speed is undesirable as a slower shutter speed causes the moving UAV's captured images to become blurry. As we change the focal length of the camera from the algorithm described in the above section, we will be inadvertently changing the f-number as well, which will change the brightness and shutter speed. To compensate for this, we can either change the ISO of the camera or simply use image processing to account for the small light changes required whenever the focal length is changed. Changing the ISO changes the image sensor's level of light sensitivity. This will mitigate any concerns with having too long of a shutter speed. However, note that the more light-sensitive the image sensor is, the lower the resolution the picture will be. Therefore, it is preferred to use image processing techniques. Finally, there is are potential image quality issues associated with changing altitude. When a fixed-wing aircraft changes altitude, its nose will go up and down. Subsequently, the camera's lens will not be parallel to the surface. To compensate for this, the camera system of the UAV should have a stabilizer which helps make the lens parallel to the surface. Fortunately, it seems a number of UAV-designed cameras already account for some of the potential problems described in this section [25].

D. Compensating for Elevation Changes of the Terrain

The previous section and models on the focal length controller assumed the terrain is flat. Once the elevation of the terrain q changes, both the field of view and the resolution should change as well. Fortunately, we have already compensated for this as we assumed the UAV can accurately estimate its elevation above the ground via variable z.

VIII. Simulations

The simulation results goes here.

IX. Conclusion

The conclusion goes here.

Acknowledgment

The authors would like to thank A, B, and C for making the research conducted in this paper possible.

Appendix
Special Case in CGTSP Tour Generation

If adjacent polygons have parallel lines that are of the same orientation, and if the resolution difference between the two polygons is less than the maximum resolution difference of the camera ($A(b^2_{max} - b^2_{min})$), then we connect the lines of these polygons so that they form a "loopback" path pattern to further minimize the number of rotations. This "loopback" path pattern can be seen in Figure 4.

Sometimes a large group of polygons of similar altitude and same orientation are next to each other, but only subsets of this large group of polygons can be grouped together to use the "loopback" path pattern. These subsets can also overlap with each other. To solve this problem, we use an algorithm similar to the connected components algorithm. For this algorithm, we define a graph, $G_2 = (V_2, E_2)$, where V_2 are the polygons, and E_2 is the set of edges (v_{2_1}, v_{2_2}) that represent the adjacency of the polygons. Then, we reiteratively use a connected components algorithm in order to find the largest connected component, T, where components are only considered "connected" if they satisfy the two conditions which allow the loopback path pattern to be done (orientation and resolution threshold). Then, we take this largest connected component out of the graph, use the "loopback" path pattern to connect the vertices (lines) in the polygons with updated edge weights for the original graph G, and then continue this connected components algorithm with $G_2 - T_1$, then with $G - T_2$, and so on until completion.

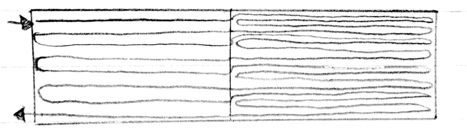
Fig. 4. An example of the loopback pattern. The two rectangles represent two sub-regions inside an MSA polygon with different resolutions. However, these two resolutions can both be obtained purely via changing the focal length of the UAV's camera.

References

[1] H. Choset, "Coverage for robotics–a survey of recent results," *Annals of mathematics and artificial intelligence*, vol. 31, no. 1-4, pp. 113–126, 2001.

[2] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1258–1276, 2013.

[3] A. Ahmadzadeh, J. Keller, G. Pappas, A. Jadbabaie, and V. Kumar, "An optimization-based approach to time-critical cooperative surveillance and coverage with uavs," in *Experimental Robotics*. Springer, 2008, pp. 491–500.

[4] C. Di Franco and G. Buttazzo, "Coverage path planning for uavs photogrammetry with energy and resolution constraints," *Journal of Intelligent & Robotic Systems*, pp. 1–18, 2016.

[5] E. U. Acar, H. Choset, Y. Zhang, and M. Schervish, "Path planning for robotic demining: Robust sensor-based coverage of unstructured environments and probabilistic methods," *The International journal of robotics research*, vol. 22, no. 7-8, pp. 441–466, 2003.

[6] E. Galceran, R. Campos, N. Palomeras, D. Ribas, M. Carreras, and P. Ridao, "Coverage path planning with real-time replanning and surface reconstruction for inspection of three-dimensional underwater structures using autonomous underwater vehicles," *Journal of Field Robotics*, vol. 32, no. 7, pp. 952–983, 2015.

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

[8] J. Hess, M. Beinhofer, and W. Burgard, "A probabilistic approach to high-confidence cleaning guarantees for low-cost cleaning robots," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 5600–5605.

[9] I. Hameed, D. Bochtis, and C. A. G. Sørensen, "An optimized field coverage planning approach for navigation of agricultural robots in fields involving obstacle areas," *International Journal of Advanced Robotic Systems*, vol. 10, no. 231, pp. 1–9, 2013.

[10] W. H. Huang, "Optimal line-sweep-based decompositions for coverage algorithms," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 1. IEEE, 2001, pp. 27–32.

[11] E. Frew, T. McGee, Z. Kim, X. Xiao, S. Jackson, M. Morimoto, S. Rathinam, J. Padial, and R. Sengupta, "Vision-based road-following using a small autonomous aircraft," in *Aerospace Conference, 2004. Proceedings. 2004 IEEE*, vol. 5. IEEE, 2004, pp. 3006–3015.

[12] S. Bochkarev and S. L. Smith, "On minimizing turns in robot coverage path planning."

[13] S. A. Sadat, J. Wawerla, and R. T. Vaughan, "Recursive non-uniform coverage of unknown terrains for uavs," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 1742–1747.

[14] S. Carpin, D. Burch, N. Basilico, T. H. Chung, and M. Kölsch, "Variable resolution search with quadrotors: Theory and practice," *Journal of Field Robotics*, vol. 30, no. 5, pp. 685–701, 2013.

[15] E. Galceran and M. Carreras, "Planning coverage paths on bathymetric maps for in-detail inspection of the ocean floor," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 4159–4164.

[16] M. Schwager, B. J. Julian, M. Angermann, and D. Rus, "Eyes in the sky: Decentralized control for the deployment of robotic camera networks," *Proceedings of the IEEE*, vol. 99, no. 9, pp. 1541–1561, 2011.

[17] X. Yu, "Optimization approaches for a dubins vehicle in coverage planning problem and traveling salesman problems," Ph.D. dissertation, Auburn University, 2015.

[18] C. A. Sugar and G. M. James, "Finding the number of clusters in a dataset," *Journal of the American Statistical Association*, 2011.

[19] J. H. Ward Jr, "Hierarchical grouping to optimize an objective function," *Journal of the American statistical association*, vol. 58, no. 301, pp. 236–244, 1963.

[20] K. Helsgaun, "Solving the equality generalized traveling salesman problem using the lin–kernighan–helsgaun algorithm," *Mathematical Programming Computation*, vol. 7, no. 3, pp. 269–287, 2015.

[21] ——, "Solving the clustered traveling salesman problem using the lin-

kernighan-helsgaun algorithm," *Computer Science Research Report*, no. 142, pp. 1–16, 2011.

[22] C. E. Noon and J. C. Bean, "An efficient transformation of the generalized traveling salesman problem," *INFOR: Information Systems and Operational Research*, vol. 31, no. 1, pp. 39–44, 1993.

[23] D. Eynard, P. Vasseur, C. Demonceaux, and V. Frémont, "Uav altitude estimation by mixed stereoscopic vision," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, 2010, pp. 646–651.

[24] A. Cherian, J. Andersh, V. Morellas, N. Papanikolopoulos, and B. Mettler, "Autonomous altitude estimation of a uav using a single onboard camera," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2009, pp. 3900–3905.

[25] Imperx, "Aerial cameras - aerial mapping cameras for a variety of industries," 2014, [Online; accessed 11-October-2016]. [Online]. Available: http://www.imperx.com/aerial-cameras/